**JS面试题**

**目录**

1. 什么是强类型语言和弱类型语言？  ............................................................................................................. 1
2. 脚本语言和编译语言的区别是什么？JS属于哪种语言类型？ ............................................................ 1
3. 什么是严格模式与混杂模式? ..................................................................................................................... 1
4. 说说严格模式的限制和优点？什么时候考虑使用严格模式开发程序？ ............................................ 1
5. defer和async的作用和区别？ ............................................................................................................... 1
6. 在JS中有哪些数据类型？ ......................................................................................................................... 1
7. 基本类型和引用类型有什么区别？它们的存储原理是什么？ ............................................................. 1
8. 在JS中进行数据类型转换的方式有哪些？ ............................................................................................ 1
9. 如何判断一个变量的数据类型？ ............................................................................................................... 1
10. 在JS中有哪几类运算符？ .......................................................................................................................... 1
11. 运算符的优先级? .......................................................................................................................................... 1
12. == 和 === 的区别是什么？ .................................................................................................................. 1
13. i++ 和 ++i 有什么不同？ ....................................................................................................................... 1
14. 什么是局部变量和全局变量？在JS中最大的全局变量是谁？ ........................................................... 1
15. 作用域链 ........................................................................................................................................................ 1
16. JS中内置了哪些常用的对象？ .................................................................................................................. 1
17. string对象方法与属性 ............................................................................................................................... 1
18. math对象方法与属性 ................................................................................................................................ 1
19. array对象方法与属性 ................................................................................................................................. 1
20. number对象方法与属性 ............................................................................................................................ 1
21. date对象方法与属性 ........................................................................................................................... 1
22. regexp对象方法与属性 ...................................................................................................................... 1
23. javascript对象有几种创建方式？ .................................................................................................... 1
24. javascript继承的6种方法以及优点和缺点 ................................................................................... 1
25. 原型链 ..................................................................................................................................................... 1
26. call() 和 apply() 的区别和作用？ ...................................................................................................... 1
27. Function的主要作用是什么？ .......................................................................................................... 1
28. 闭包 ......................................................................................................................................................... 1
29. 回调 ......................................................................................................................................................... 1
30. 匿名 ......................................................................................................................................................... 1
31. 递归 ......................................................................................................................................................... 1
32. DOM的主要作用是什么？ ................................................................................................................. 1
33. DOM操作 —— 如何添加、移除、移动、复制、创建和查找节点等。 .................................... 1
34. BOM的主要作用是什么？ .................................................................................................................... 1
35. 什么是同步和异步？ ............................................................................................................................... 1
36. 请解释一下 JavaScript 的同源策略。 ........................................................................................... 1
37. 什么是Ajax？ ...................................................................................................................................... 1
38. 创建Ajax的过程？ ............................................................................................................................. 1
39. ajax的缺点和在IE下的问题？ ......................................................................................................... 1
40. HTTP状态码 ......................................................................................................................................... 1
41. XML和JSON的区别？ .................................................................................................................... 1
42. 如何解决跨域问题？ .......................................................................................................................... 1
43. 请描述一下 cookies，sessionStorage 和 localStorage 的区别？ .................................. 1
44. 如何删除一个cookie？ .................................................................................................................... 1
45. 什么是web worker，及如何使用？ .............................................................................................. 1
46. Audio和video对象常用的属性和方法有哪些？ ....................................................................... 1
47. 如何配置离线存储？ .......................................................................................................................... 1
48. 什么是事件委托？ .............................................................................................................................. 1
49. ES6有哪些新增特性？ ...................................................................................................................... 1
50. 什么是强类型语言和弱类型语言？
51. **强类型语言：**强类型语言也称为强类型定义语言。是一种总是强制类型定义的语言，要求变量的使用要严格符合定义，所有变量都必须先定义后使用。也就是说，一旦一个变量被指定了某个数据类型，如果不经过[强制转换](http://baike.baidu.com/item/%E5%BC%BA%E5%88%B6%E8%BD%AC%E6%8D%A2/4463315" \t "http://baike.baidu.com/_blank)，那么它就永远是这个数据类型了。
52. **弱类型语言：**数据类型可以被忽略的语言。它与强类型定义语言相反, 一个变量可以赋不同数据类型的值。
53. 脚本语言和编译语言的区别是什么？JS属于哪种语言类型？
54. **脚本语言：**[解释性语言](https://www.baidu.com/s?wd=%E8%A7%A3%E9%87%8A%E6%80%A7%E8%AF%AD%E8%A8%80&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YYrjPWnHm1P1R4mhn3P10s0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHT4PW0YPHRYnWTYPjb4rjfvPs" \t "https://zhidao.baidu.com/question/_blank)中，用于执行简单任务的语言。一般没有 类 或者此类高级功能。
55. **编译语言：**在运行前需要将源代码转换为可执行代码的语言。

**JS属于脚本语言**

1. 什么是严格模式与混杂模式?

**严格模式：**又称标准模式，是指浏览器按照 W3C 标准解析代码。 **混杂模式：**又称怪异模式或兼容模式，是指浏览器用自己的方式解析代码。 **如何区分：**浏览器解析时到底使用严格模式还是混杂模式，与网页中的 DTD 直接相关。

1. 说说严格模式的限制和优点？什么时候考虑使用严格模式开发程序？

**限制:**

1. 在正常模式中，如果一个变量没有声明就赋值，默认是全局变量。严格模式禁止这种用法，全局变量必须显式声明。
2. 严格模式对动态绑定做了一些限制。某些情况下，只允许静态绑定。也就是说，属性和方法到底归属哪个对象，在编译阶段就确定。这样做有利于编译效率的提高，也使得代码更容易阅读，更少出现意外。

**优点：**

1. 消除Javascript语法的一些不合理、不严谨之处，减少一些怪异行为;
2. 消除代码运行的一些不安全之处，保证代码运行的安全；
3. 提高编译器效率，增加运行速度；
4. 为未来新版本的Javascript做好铺垫。

**什么时候考虑使用严格模式：**

"严格模式"体现了Javascript更合理、更安全、更严谨的发展方向，另一方面，同样的代码，在"严格模式"中，可能会有不一样的运行结果；一些在"正常模式"下可以运行的语句，在"严格模式"下将不能运行。因此，在开发大型项目时，或多人合作的项目时，使用严格模式是更好的选择。

1. defer和async的作用和区别？

**async模式**

<script type="text/javascript" src="x.min.js" async="async"></script>

当浏览器遇到 script 标签时，文档的解析不会停止，其他线程将下载脚本，脚本下载完成后开始执行脚本，脚本执行的过程中文档将停止解析，直到脚本执行完毕。

**defer模式**

<script type="text/javascript" src="x.min.js" defer="defer"></script>

当浏览器遇到 script 标签时，文档的解析不会停止，其他线程将下载脚本，待到文档解析完成，脚本才会执行。

1. 在JS中有哪些数据类型？

**基本类型：**String，Number，Null，Underfine，boolean

**引用类型：**object

1. 基本类型和引用类型有什么区别？它们的存储原理是什么？

**基本数据类型**在被创建时，在栈上给其划分一块内存，将数值直接存储在栈上；

**引用数据类型**在被创建时，首先要在栈上给其引用（句柄）分配一块内存，而对象的具体信息都存储在堆内存上，然后由栈上面的引用指向堆中对象的地址。

1. 在JS中进行数据类型转换的方式有哪些？

parseInt()和parseFloat()两个转换函数。前者把值转换成整数，后者把值转换成浮点数。只有对String类型调用这些方法，这两个函数才能正确运行。

3种强制类型转换：

Boolean(value) —— 把给定的值转换成Boolean型。

Number(value) —— 把给定的值转换成数字（可以是整数或浮点数）；

String(value) —— 把给定的值转换成字符串。

1. 如何判断一个变量的数据类型？

**typeof：**用于返回基本数据类型

**Instanceof：**判断已知对象类型的方法

注意：instanceof 后面一定要是对象类型，并且大小写不能错，该方法适合一些条件选择或分支

**constructor：**根据对象的constructor判断

注意： constructor 在类继承时会出错

1. 在JS中有哪几类运算符？
2. [算术运算符](http://www.jb51.net/article/56342.htm" \l "arithmetic)
3. [赋值运算符](http://www.jb51.net/article/56342.htm" \l "assign)
4. [比较运算符](http://www.jb51.net/article/56342.htm" \l "compare)
5. [三元运算符](http://www.jb51.net/article/56342.htm" \l "ternary)
6. [逻辑运算符](http://www.jb51.net/article/56342.htm" \l "logic)
7. [字符串连接运算符](http://www.jb51.net/article/56342.htm" \l "concatenation)
8. 运算符的优先级?

下面的表将所有运算符按照优先级的不同从高到低排列。

|  |  |  |  |
| --- | --- | --- | --- |
| **优先级** | **运算类型** | **关联性** | **运算符** |
| 19 | [圆括号](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Grouping" \o "圆括号运算符( ) 用来控制表达式中的运算优先级.) | n/a | ( … ) |
| 18 | [成员访问](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Property_Accessors" \l "%E7%82%B9%E7%AC%A6%E5%8F%B7%E8%A1%A8%E7%A4%BA%E6%B3%95" \o "属性访问器提供了两种方式用于访问一个对象的属性，它们分别是点符号和括号。) | 从左到右 | . |
| [需计算的成员访问](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Property_Accessors" \l "%E6%8B%AC%E5%8F%B7%E8%A1%A8%E7%A4%BA%E6%B3%95" \o "属性访问器提供了两种方式用于访问一个对象的属性，它们分别是点符号和括号。) | 从左到右 | [ … ] |
| [new](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/new" \o "(new operator) 新运算符创建一个用户定义的对象类型的实例或具有构造函数的内置对象类型之一。) (带参数列表) | n/a | new ( … ) |
| 17 | [函数调用](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Guide/Functions" \o "JavaScript/Reference/Operators/Special_Operators/function_call) | 从左到右 | ( … ) |
| [new](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/new" \o "JavaScript/Reference/Operators/Special_Operators/new_Operator) (无参数列表) | 从右到左 | new |
| 16 | [后置递增](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators" \l "Increment" \o "JavaScript/Reference/Operators/Arithmetic_Operators)(运算符在后) | n/a | ++ |
| [后置递减](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators" \l "Decrement" \o "JavaScript/Reference/Operators/Arithmetic_Operators)(运算符在后) | n/a | -- |
| 15 | [逻辑非](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Logical_Operators" \l "Logical_NOT) | 从右到左 | ! |
| [按位非](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators" \l "Bitwise_NOT" \o "JavaScript/Reference/Operators/Bitwise_Operators) | 从右到左 | ~ |
| [一元加法](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators" \l "Unary_plus" \o "JavaScript/Reference/Operators/Arithmetic_Operators) | 从右到左 | + |
| [一元减法](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators" \l "Unary_negation" \o "JavaScript/Reference/Operators/Arithmetic_Operators) | 从右到左 | - |
| [前置递增](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators" \l "Increment" \o "JavaScript/Reference/Operators/Arithmetic_Operators) | 从右到左 | ++ |
| [前置递减](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators" \l "Decrement" \o "JavaScript/Reference/Operators/Arithmetic_Operators) | 从右到左 | -- |
| [typeof](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/typeof" \o "JavaScript/Reference/Operators/Special_Operators/typeof_Operator) | 从右到左 | typeof |
| [void](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/void" \o "JavaScript/Reference/Operators/Special_Operators/void_Operator) | 从右到左 | void |
| [delete](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/delete" \o "JavaScript/Reference/Operators/Special_Operators/delete_Operator) | 从右到左 | delete |
| 14 | [乘法](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators" \l "Multiplication" \o "JavaScript/Reference/Operators/Arithmetic_Operators) | 从左到右 | \* |
| [除法](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators" \l "Division" \o "JavaScript/Reference/Operators/Arithmetic_Operators) | 从左到右 | / |
| [取模](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators" \l "Remainder" \o "JavaScript/Reference/Operators/Arithmetic_Operators) | 从左到右 | % |
| 13 | [加法](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators" \l "Addition" \o "JavaScript/Reference/Operators/Arithmetic_Operators) | 从左到右 | + |
| [减法](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators" \l "Subtraction" \o "JavaScript/Reference/Operators/Arithmetic_Operators) | 从左到右 | - |
| 12 | [按位左移](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators" \o "JavaScript/Reference/Operators/Bitwise_Operators) | 从左到右 | << |
| [按位右移](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators" \o "JavaScript/Reference/Operators/Bitwise_Operators) | 从左到右 | >> |
| [无符号右移](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators" \o "JavaScript/Reference/Operators/Bitwise_Operators) | 从左到右 | >>> |
| 11 | [小于](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Comparison_Operators" \l "Less_than_operator" \o "JavaScript/Reference/Operators/Comparison_Operators) | 从左到右 | < |
| [小于等于](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Comparison_Operators" \l "Less_than__or_equal_operator" \o "JavaScript/Reference/Operators/Comparison_Operators) | 从左到右 | <= |
| [大于](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators" \l "Greater_than_operator" \o "JavaScript/Reference/Operators/Comparison_Operators) | 从左到右 | > |
| [大于等于](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Comparison_Operators" \l "Greater_than_or_equal_operator" \o "JavaScript/Reference/Operators/Comparison_Operators) | 从左到右 | >= |
| [in](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/in" \o "JavaScript/Reference/Operators/Special_Operators/in_Operator) | 从左到右 | in |
| [instanceof](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/instanceof" \o "JavaScript/Reference/Operators/Special_Operators/instanceof_Operator) | 从左到右 | instanceof |
| 10 | [等号](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Comparison_Operators" \l "Equality" \o "JavaScript/Reference/Operators/Comparison_Operators) | 从左到右 | == |
| [非等号](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Comparison_Operators" \l "Inequality" \o "JavaScript/Reference/Operators/Comparison_Operators) | 从左到右 | != |
| [全等号](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Comparison_Operators" \l "Identity" \o "JavaScript/Reference/Operators/Comparison_Operators) | 从左到右 | === |
| [非全等号](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Comparison_Operators" \l "Nonidentity" \o "JavaScript/Reference/Operators/Comparison_Operators) | 从左到右 | !== |
| 9 | [按位与](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators" \l "Bitwise_AND" \o "JavaScript/Reference/Operators/Bitwise_Operators) | 从左到右 | & |
| 8 | [按位异或](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators" \l "Bitwise_XOR" \o "JavaScript/Reference/Operators/Bitwise_Operators) | 从左到右 | ^ |
| 7 | [按位或](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators" \l "Bitwise_OR" \o "JavaScript/Reference/Operators/Bitwise_Operators) | 从左到右 | | |
| 6 | [逻辑与](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Logical_Operators" \l "Logical_AND" \o "JavaScript/Reference/Operators/Logical_Operators) | 从左到右 | && |
| 5 | [逻辑或](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Logical_Operators" \l "Logical_OR" \o "JavaScript/Reference/Operators/Logical_Operators) | 从左到右 | || |
| 4 | [条件运算符](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Conditional_Operator" \o "JavaScript/Reference/Operators/Special_Operators/Conditional_Operator) | 从右到左 | ? … : … |
| 3 | [赋值](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Assignment_Operators" \o "JavaScript/Reference/Operators/Assignment_Operators) | 从右到左 | = |
| += |
| -= |
| \*= |
| /= |
| %= |
| <<= |
| >>= |
| >>>= |
| &= |
| ^= |
| |= |
| 2 | [yield](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/yield" \o "JavaScript/Reference/Operators/yield) | 从右到左 | yield |
| [yield\*](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/yield*" \o "JavaScript/Reference/Operators/yield) | 从右到左 | yield\* |
| 1 | [展开运算符](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Spread_operator" \o "JavaScript/Reference/Operators/Spread_operator) | n/a | ... |
| 0 | [逗号](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Comma_Operator" \o "JavaScript/Reference/Operators/Comma_Operator) | 从左到右 | , |

1. == 和 === 的区别是什么？

==判断值是否相等,在做判断时，js会默认进行类型转换

===判断值及类型是否完全相等，不会进行类型转换

1. i++ 和 ++i 有什么不同？

根本区别是语义上的区别，一个返回 + 之后的值一个返回 + 之前的值。

如果没有用到返回值的话，区别在于效率。

若i是内置的数值类型，两者完全一样

若i是一些自定义的类，如iterator，++i的效率 > = i++的效率

对于后者推荐都用++i；对于前者，用哪个是程序风格问题，i++的好处是更符合人类思维习惯，++i的好处是每次都用这种形式就不用考虑i的类型。

1. 什么是局部变量和全局变量？在JS中最大的全局变量是谁？

全局变量和局部变量的区别如下：

1. 作用域不同：全局变量的作用域为整个程序，而局部变量的作用域为当前函数或循环等

2. 内存存储方式不同：全局变量存储在全局数据区中，局部变量存储在栈区

3. 生命期不同：全局变量的生命期和主程序一样，随程序的销毁而销毁，局部变量在函数内部或循环内部，随函数的退出或循环退出就不存在了

4. 使用方式不同：全局变量在声明后程序的各个部分都可以用到，但是局部变量只能在局部使用。函数内部会优先使用局部变量再使用全局变量

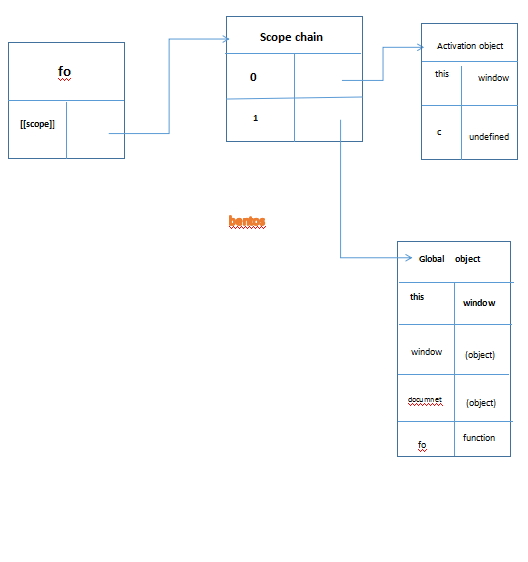
在js中自大的全局变量：window

1. 作用域链

在JavaScript中，函数也是对象，实际上，JavaScript里一切都是对象。函数对象和其它对象一样，拥有可以通过代码访问的属性和一系列仅供JavaScript引擎访问的内部属性。其中一个内部属性是[[Scope]]，由ECMA-262标准第三版定义，该内部属性包含了函数被创建的作用域中对象的集合，这个集合被称为函数的作用域链，它决定了哪些数据能被函数访问。

执行函数时会创建一个称为“运行期上下文”的内部对象，运行期上下文定义了函数执行时的环境。每个运行期上下文都有自己的作用域链，用于标识符解析，当运行期上下文被创建时，而它的作用域链初始化为当前运行函数的[[Scope]]所包含的对象。

　　这些值按照它们出现在函数中的顺序被复制到运行期上下文的作用域链中。它们共同组成了一个新的对象，叫“活动对象”，该对象包含了函数的所有局部变量、命名参数、参数集合以及this，然后此对象会被推入作用域链的前端，当运行期上下文被销毁，活动对象也随之销毁。新的作用域链如下图所示：



1. JS中内置了哪些常用的对象？
2. Array
3. String
4. Date
5. Math
6. Boolean
7. Number
8. Function
9. Global
10. Error
11. RegExp
12. Object
13. string对象方法与属性

属性:  
constructor 所建立对象的函数参考  
prototype    能够为对象加入的属性和方法  
length      返回字符串的字符长度

方法:  
anchor("name")用来把字符串转换为HTML锚点标记内(<A NAME=>)  
charAt(index) 返回指定索引处的字符  
charCodeAt(index)返回一个整数,该整数表示String对象中指定位置处的字符的Unicode编码  
concat(string2)连接两条或多条字符串   
indexOf(searchString, startIndex) 返回字符串中第一个出现指定字符串的位置  
lastlndexOf(searchString, startIndex) 返回字符串中最后一个出现指定字符串的位置  
match(regex) 在字符串中查找指定值  
replace(regex, newString)将字符串中的某些字符替换成其它字符  
search(regex) 针对某执行值对字符串进行查找  
slice(startIndex, endIndex)将部分字符抽出并在新的字符串中返回剩余部分  
split(delimiter)将字符串分配为数组  
substr(startIndex, length) 从startIndex取,取length个字符  
substring(startIndex, endIndex) 从startIndex和endIndex之间的字符,不包括endIndex

toLowerCase() 把字符串中的文本变成小写  
toUpperCase() 把字符串中的文本变成大写  
toSource()    显示对象的源代码  
valueOf()    返回字符串对象的原始值

1. math对象方法与属性

**属性:**  
constructor      所建立对象的函数参考   
prototype       能够为对象加入的属性和方法  
E           欧拉常量,自然对数的底(约等于2.718)  
LN2          2的自然对数(约等于0.693)  
LN10          10的自然对数(约等于2.302)  
LOG2E 以2为底的e的对数.(约等于1.442)  
LOG10E         以10为底的e的对数(约等于0.434)  
PI ∏的值(约等于3.14159)  
SQRT1\_2 1/2(0.5)的平方根(即l除以2的平方根,约等于o.707)  
SQRT2 2的平方根(约等于1.414)

**方法:**  
abs(x)     返回数字的绝对值  
acos(x)    返回数字的反余弦值  
asin(x)    返回数字的反正弦值  
atan(x)    返回位于-PI/2 和 PI/2 的反正切值  
atan2(y,x) 返回（x,y）位于 -PI 到 PI 之间的角度  
ceil(x)    返回 x 四舍五入后的最大整数  
cos(x)     返回一个数字的余弦值  
exp(x)     返回 E^x 值  
floor(x)    返回 x 四舍五入后的最小整数  
log(x)     返回底数为E的自然对数  
max(x,y)    返回 x 和 y 之间较大的数  
min(x,y)    返回 x 和 y 之间较小的数  
pow(x,y)    返回 y^x 的值  
random()    返回位于 0 到 1 之间的随机函数  
round(x)    四舍五入后取整  
sin(x) 返回数字的正弦值  
sqrt(x)    返回数字的平方根  
tan(x)     返回一个角度的正切值  
toSource() 显示对象的源代码  
valueOf() 返回数学对象的原始值

1. array对象方法与属性

**属性:**  
constructor 所建立对象的函数参考  
prototype    能够为对象加入的属性和方法  
length      获取数组元素的个数,即最大下标加1

**方法(13):**  
concat(array1,arrayn)将两个或两个以上的数组值连接起来,合并后返回结果  
join(string) 将数组中元素合并为字符串,string为分隔符.如省略参数则直接合并,不再分隔  
pop()      移除数组中的最后一个元素并返回该元素  
push(value) 在数组的末尾加上一个或多个元素,并且返回新的数组长度值  
reverse()    颠倒数组中元素的顺序,反向排列  
shift() 移除数组中的第一个元素并返回该元素  
slice(start, deleteCount, [item1[, item2[,...[,itemN]]]]) 返从一个数组中移除一个或多个元素,如果必要,在所移除元素的位置上插入新元素,返回所移除的元素  
sort(compare Function) 在未指定排序号的情况下,按照元素的字母顺序排列,如果不是字符串类型则转换成字符串再排序,返回排序后的数组  
splice()     为数组删除并添加新的元素  
toSource()    显示对象的源代码  
toString() 将数组所有元素返回一个字符串,其间用逗号分隔  
unshift(value)为数组的开始部分加上一个或多个元素,并且返回该数组的新长度  
valueOf()    返回数组对象的原始值

1. number对象方法与属性

方法:  
toString       Returns a string representing the specified object   
valueOf()      返回数学对象的原始值

1. date对象方法与属性

**属性:**  
constructor      所建立对象的函数参考  
prototype       能够为对象加入的属性和方法

方法:  
getDay()        返回一周中的第几天(0-6)  
getYear()       返回年份.2000年以前为2位,2000(包含)以后为4位  
getFullYear()     返回完整的4位年份数  
getMonth()      返回月份数(0-11)  
getDate()       返回日(1-31)  
getHours()       返回小时数(0-23)  
getMinutes()      返回分钟(0-59)  
getSeconds()      返回秒数(0-59)  
getMilliseconds() 返回毫秒(0-999)  
getUTCDay()      依据国际时间来得到现在是星期几(0-6)  
getUTCFullYear()    依据国际时间来得到完整的年份  
getUTCMonth()     依据国际时间来得到月份(0-11)  
getUTCDate()      依据国际时间来得到日(1-31)  
getUTCHours()     依据国际时间来得到小时(0-23)  
getUTCMinutes()    依据国际时间来返回分钟(0-59)  
getUTCSeconds()    依据国际时间来返回秒(0-59)  
getUTCMilliseconds()依据国际时间来返回毫秒(0-999)  
getTime()       返回从1970年1月1号0:0:0到现在一共花去的毫秒数  
getTimezoneoffset() 返回时区偏差值,即格林威治平均时间(GMT)与运行脚本的计算机所处时区设置之间相差的分钟数)  
parse(dateString)   返回在Date字符串中自从1970年1月1日00:00:00以来的毫秒数  
setYear(yearInt)    设置年份.2位数或4位数  
setFullYear(yearInt)设置年份.4位数  
setMonth(monthInt) 设置月份(0-11)  
setDate(dateInt)    设置日(1-31)  
setHours(hourInt) 设置小时数(0-23)  
setMinutes(minInt) 设置分钟数(0-59)  
setSeconds(secInt) 设置秒数(0-59)  
setMilliseconds(milliInt) 设置毫秒(0-999)  
setUTCFullYear(yearInt) 依据国际时间来设置年份  
setUTCMonth(monthInt) 依据国际时间来设置月(0-11)  
setUTCDate(dateInt)     依据国际时间来设置日(1-31)  
setUTCHours(hourInt)    依据国际时间来设置小时  
setUTCMinutes(minInt) 依据国际时间来设置分钟  
setUTCSeconds(secInt)    依据国际时间来设置秒  
setUTCMilliseconds(milliInt)依据国际时间来设置毫秒  
setTime(timeInt)    设置从1970年1月1日开始的时间.毫秒数  
toGMTString()     根据格林威治时间将Date对象的日期(一个数值)转变成一个GMT时间字符串,如:Weds,15 June l997 14:02:02 GMT  
toUTCString()     根据通用时间将一个Date对象的日期转换为一个字符串  
toLocaleString()    把Date对象的日期(一个数值)转变成一个字符串,使用所在计算机上配置使用的特定日期格式  
toSource()       显示对象的源代码  
toString()       将日期对象转换为字符串  
UTC(yyyy, mm, dd, hh, mm, ss, msec)返回从格林威治标准时间到指定时间的差距,单位为毫秒  
valueOf()       返回日期对象的原始值

1. regexp对象方法与属性

**属性：**

multiline 对象是否具有标志 m。

ignoreCase 对象是否具有标志 i。

global 对象是否具有标志 g。

**方法：**

compile(regexp,modifier) 方法也可用于改变和重新编译正则表达式

exec(string) 方法用于检索字符串中的正则表达式的匹配。

test(string) 方法用于检测一个字符串是否匹配某个模式.

stringObject.search(regexp) 用于检索字符串中指定的子字符串，或检索与正则表达式相匹配的子字符串。

stringObject.replace(regexp/substr,replacement) 用于在字符串中用一些字符替换另一些字符，或替换一个与正则表达式匹配的子串。

stringObject.split(separator,howmany) 方法用于把一个字符串分割成字符串数组。

1. javascript对象有几种创建方式？
2. 使用Object构造函数来创建一个对象

var person = new Object();

person.name="kevin";

person.age=31;

alert(person.name);

alert(person["name"])

1. 使用对象字面量创建一个对象

var person = {

name:"Kevin",

age:31,

5:"Test"

};

alert(person.name);

alert(person["5"]);

1. 使用工厂模式创建对象

function createPerson(name, age,job)

{

var o = new Object();

o.name=name;

o.age=31;

o.sayName=function()

{

alert(this.name);

};

return o;

}

createPerson("kevin",31,"se").sayName();

1. 使用自定义构造函数模式创建对象

function Person(name,age,job)

{

this.name=name;

this.age=age;

this.job=job;

this.sayName=function()

{

alert(this.name);

};

}

var person = new Person("kevin",31,"SE");

person.sayName();

1. 使用原型模式创建对象

function Person(){}

Person.prototype = {

constructor : Person,

name:"kevin",

age:31,

job:"SE",

friends:["Jams","Martin"],

sayFriends:function()

{

alert(this.friends);

}

};

var person1 = new Person();

person1.friends.push("Joe");

person1.sayFriends();

var person2 = new Person();

person2.sayFriends();

1. 组合使用原型模式和构造函数创建对象

function Person(name,age,job){

this.name=name;

this.age=age;

this.job=job;  
　　 this.friends=["Jams","Martin"];

}

Person.prototype.sayFriends=function(){

alert(this.friends);

};

var person1 = new Person("kevin",31,"SE");

var person2 = new Person("Tom",30,"SE");

person1.friends.push("Joe");

person1.sayFriends();

person2.sayFriends();

1. 动态原型模式；

function Person(name,age,job){

//属性

this.name=name;

this.age=age;

this.job=job;

this.friends=["Jams","Martin"];

//方法

if(typeof this.sayName !="function"){

Person.prototype.sayName=function() {

alert(this.name);

};

Person.prototype.sayFriends=function(){

alert(this.friends);

};

}

}

var person = new Person("kevin",31,"SE");

person.sayName();

person.sayFriends();

1. 寄生构造函数模式
2. 稳妥构造函数模式
3. javascript继承的6种方法以及优点和缺点
4. **简单原型链**

**具体实现**

function Super(){

    this.val = 1;

    this.arr = [1];

}

function Sub(){

    // ...

}

Sub.prototype = new Super();    // 核心

var sub1 = new Sub();

var sub2 = new Sub();

sub1.val = 2;

sub1.arr.push(2);

alert(sub1.val);    // 2

alert(sub2.val);    // 1

alert(sub1.arr);    // 1, 2

alert(sub2.arr);    // 1, 2

**核心**

拿父类实例来充当子类原型对象

**优缺点**

**优点：**

1. 简单，易于实现

**缺点：**

1. 修改sub1.arr后sub2.arr也变了，因为来自原型对象的引用属性是所有实例共享的。
2. 可以这样理解：执行sub1.arr.push(2);先对sub1进行属性查找，找遍了实例属性（在本例中没有实例属性），没找到，就开始顺着原型链向上找，拿到了sub1的原型对象，一搜身，发现有arr属性。于是给arr末尾插入了2，所以sub2.arr也变了
3. 创建子类实例时，无法向父类构造函数传参
4. **借用构造函数**

function Super(val){

    this.val = val;

    this.arr = [1];

    this.fun = function(){

        // ...

    }

}

function Sub(val){

    Super.call(this, val);   // 核心

    // ...

}

var sub1 = new Sub(1);

var sub2 = new Sub(2);

sub1.arr.push(2);

alert(sub1.val);    // 1

alert(sub2.val);    // 2

alert(sub1.arr);    // 1, 2

alert(sub2.arr);    // 1

alert(sub1.fun === sub2.fun);   // false

**核心**

借父类的构造函数来增强子类实例，等于是把父类的实例属性复制了一份给子类实例装上了（完全没有用到原型）

**优缺点**

**优点：**

1. 解决了子类实例共享父类引用属性的问题
2. 创建子类实例时，可以向父类构造函数传参

**缺点：**

1. 无法实现函数复用，每个子类实例都持有一个新的fun函数，太多了就会影响性能，内存爆炸。
2. **组合继承（最常用）**

function Super(){

    // 只在此处声明基本属性和引用属性

    this.val = 1;

    this.arr = [1];

}

//  在此处声明函数

Super.prototype.fun1 = function(){};

Super.prototype.fun2 = function(){};

//Super.prototype.fun3...

function Sub(){

    Super.call(this);   // 核心

    // ...

}

Sub.prototype = new Super();    // 核心

var sub1 = new Sub(1);

var sub2 = new Sub(2);

alert(sub1.fun === sub2.fun);   // true

**核心**

把实例函数都放在原型对象上，以实现函数复用。同时还要保留借用构造函数方式的优点，通过Super.call(this);继承父类的基本属性和引用属性并保留能传参的优点；通过Sub.prototype = new Super();继承父类函数，实现函数复用

**优缺点**

**优点：**

1. 不存在引用属性共享问题
2. 可传参
3. 函数可复用

**缺点:**

（一点小瑕疵）子类原型上有一份多余的父类实例属性，因为父类构造函数被调用了两次，生成了两份，而子类实例上的那一份屏蔽了子类原型上的。。。又是内存浪费，比刚才情况好点，不过确实是瑕疵

1. **寄生组合继承（最佳方式）**

function beget(obj){   // 生孩子函数 beget：龙beget龙，凤beget凤。

    var F = function(){};

    F.prototype = obj;

    return new F();

}

function Super(){

    // 只在此处声明基本属性和引用属性

    this.val = 1;

    this.arr = [1];

}

//  在此处声明函数

Super.prototype.fun1 = function(){};

Super.prototype.fun2 = function(){};

//Super.prototype.fun3...

function Sub(){

    Super.call(this);   // 核心

    // ...

}

var proto = beget(Super.prototype); // 核心

proto.constructor = Sub;            // 核心

Sub.prototype = proto;              // 核心

var sub = new Sub();

alert(sub.val);

alert(sub.arr);

**核心**

用beget(Super.prototype);切掉了原型对象上多余的那份父类实例属性

**优缺点**

优点：完美了

1. **原型式**

function beget(obj){   // 生孩子函数 beget：龙beget龙，凤beget凤。

     var F = function(){};

     F.prototype = obj;

     return new F();

}

function Super(){

    this.val = 1;

    this.arr = [1];

}

// 拿到父类对象

var sup = new Super();

// 生孩子

var sub = beget(sup);   // 核心

// 增强

sub.attr1 = 1;

sub.attr2 = 2;

//sub.attr3...

alert(sub.val);     // 1

alert(sub.arr);     // 1

alert(sub.attr1);   // 1

**核心**

用生孩子函数得到得到一个“纯洁”的新对象（“纯洁”是因为没有实例属性），再逐步增强之（填充实例属性）

**优缺点**

**优点：**

从已有对象衍生新对象，不需要创建自定义类型（更像是对象复制，而不是继承。。）

**缺点：**

1. 原型引用属性会被所有实例共享，因为是用整个父类对象来充当了子类原型对象，所以这个缺陷无可避
2. 无法实现代码复用（新对象是现取的，属性是现添的，都没用函数封装，怎么复用）
3. **寄生式**

function beget(obj){   // 生孩子函数 beget：龙beget龙，凤beget凤。

    var F = function(){};

    F.prototype = obj;

    return new F();

}

function Super(){

    this.val = 1;

    this.arr = [1];

}

function getSubObject(obj){

    // 创建新对象

    var clone = beget(obj); // 核心

    // 增强

    clone.attr1 = 1;

    clone.attr2 = 2;

    //clone.attr3...

    return clone;

}

var sub = getSubObject(new Super());

alert(sub.val);     // 1

alert(sub.arr);     // 1

alert(sub.attr1);   // 1

**核心**

给原型式继承穿了个马甲而已，看起来更像继承了（上面介绍的原型式继承更像是对象复制）

注意：beget函数并不是必须的，换言之，创建新对象 -> 增强 -> 返回该对象，这样的过程叫寄生式继承，新对象是如何创建的并不重要（用beget生的，new出来的，字面量现做的。。都可以）

**优缺点**

**优点：**

还是不需要创建自定义类型

**缺点：**

无法实现函数复用

1. 原型链

函数的原型对象constructor默认指向函数本身，原型对象除了有原型属性外，为了实现继承，还有一个原型链指针proto，该指针指向上一层的原型对象，而上一层的原型对象的结构依然类似，这样利用proto一直指向Object的原型对象上，而Object的原型对象用Object.proto = null表示原型链的最顶端，如此变形成了javascript的原型链继承，同时也解释了为什么所有的javascript对象都具有Object的基本方法。

1. call() 和 apply() 的区别和作用？

**作用：**

call和apply都可以让B对象调用A对象的方法或属性，并且修改了this的当前作用对象。

**区别：**

1. call，apply都属于Function.prototype的一个方法，它是JavaScript引擎内在实现的，因为属于Function.prototype，所以每个Function对象实例(就是每个方法)都有call，apply属性。既然作为方法的属性，那它们的使用就当然是针对方法的了，这两个方法是容易混淆的，因为它们的作用一样，只是使用方式不同。
2. 语法：foo.call(this, arg1,arg2,arg3) == foo.apply(this, arguments) == this.foo(arg1, arg2, arg3);
3. 相同点：两个方法产生的作用是完全一样的。
4. 不同点：方法传递的参数不同。
5. Function的主要作用是什么？
6. 代码复用
7. 通过构造函数创建对象
8. 闭包函数
9. 闭包是一个拥有许多变量和绑定了这些变量的环境的表达式（通常是一个函数），因而这些变量也是该表达式的一部分。
10. 相关连接：<http://www.jb51.net/article/24101.htm>
11. 回调函数

**定义：**

函数A作为参数(函数引用)传递到另一个函数B中，并且这个函数B执行函数A。我们就说函数A叫做回调函数。如果没有名称(函数表达式)，就叫做匿名回调函数。

相关阅读：http://www.jb51.net/article/54750.htm

**例：**

var func1=function(callback){

  //do something.

  (callback && typeof(callback) === "function") && callback();

}

func1(func2);

  var func2=function(){

}

1. 匿名函数
2. **函数字面量：**首先声明一个函数对象，然后执行它。

(function () {

alert(1);

})();

1. **优先表达式：**由于Javascript执行表达式是从圆括号里面到外面，所以用圆括号强制执行声明的函数。

(function () {

alert(2);

}());

1. **Void操作符：**用void操作符去执行一个没有用圆括号包围的一个单独操作数。

void function () {

alert(3);

}()

**匿名函数的应用**

《Javascript的一种模块模式》中的第一句话就是“全局变量是魔鬼”。配合var关键字，匿名函数可以有效的保证在页面上写入Javascript，而不会造成全局变量的污染。这在给一个不是很熟悉的页面增加Javascript时非常有效，也很优美。实际上，YUI以及其相应的范例中大量使用匿名函数，其他的Javascript库中也不乏大量使用。

1. 递归函数

一个含直接或间接调用本函数语句的函数被称之为递归函数，在上面的例子中能够看出，它必须满足以下两个条件：

1） 在每一次调用自己时，必须是（在某种意义上）更接近于解；

2） 必须有一个终止处理或计算的准则。

1. DOM的主要作用是什么？

HTML DOM 定义了所有 HTML 元素的对象和属性，以及访问它们的方法

DOM 是关于如何获取、修改、添加或删除 HTML 元素的标准。

1. DOM操作 —— 如何添加、移除、移动、复制、创建和查找节点等。

**创建新节点**

1. createDocumentFragment() //创建一个DOM片段
2. createElement\_x() //创建一个具体的元素
3. createTextNode() //创建一个文本节点

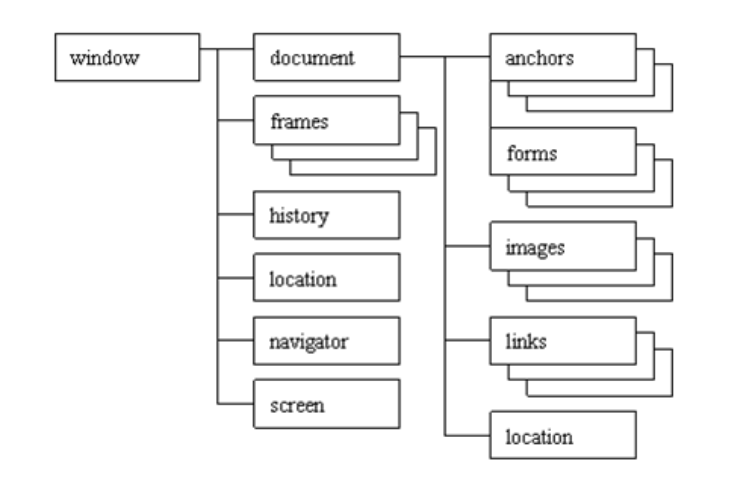
**添加、移除、替换、插入**

1. appendChild() //添加节点
2. removeChild() //移除节点
3. replaceChild() //替换节点
4. insertBefore() //插入节点

**查找**

1. getElementsByTagName() //通过标签名称
2. getElementsByName() //通过元素的Name属性的值
3. getElementById() //通过元素Id，唯一性
4. BOM的主要作用是什么？

BOM是浏览器提供给JS操作的一个对象模型，实现了JS可以通过操作BOM来实现浏览器的各种行为。



1. 什么是同步和异步？

同步就是指一个进程在执行某个请求的时候，若该请求需要一段时间才能返回信息，那么这个进程将会一直等待下去，直到收到返回信息才继续执行下去；异步是指进程不需要一直等下去，而是继续执行下面的操作，不管其他进程的状态。当有消息返回时系统会通知进程进行处理，这样可以提高执行的效率。

而我们平时经常讨论的同步问题多发生在多线程环境中的数据共享问题。即当多个线程需要访问同一个资源时，它们需要以某种顺序来确保该资源在某一特定时刻只能被一个线程所访问，如果使用异步，程序的运行结果将不可预料。因此，在这种情况下，就必须对数据进行同步，即限制只能有一个进程访问资源，其他线程必须等待。

1. 请解释一下 JavaScript 的同源策略。

相关阅读：<http://www.jb51.net/article/66500.htm>

1. 什么是Ajax？

Ajax不是一种技术。实际上，它由几种蓬勃发展的技术以新的强大方式组合而成。Ajax包含：

1. 基于XHTML和CSS标准的表示；
2. 使用Document Object Model进行动态显示和交互；
3. 使用XMLHttpRequest与服务器进行异步通信；
4. 使用JavaScript绑定一切。
5. 创建Ajax的过程？
6. 创建XMLHttpRequest对象，也就是创建一个异步调用对象；
7. 创建一个新的HTTP请求，并指定改HTTP请求的方法、URL以及验证信息；
8. 设置响应HTTP状态变化的函数；
9. 发送HTTP请求；
10. 获取异步调用返回的数据；
11. 使用javascript和DOM实现局部刷新；
12. ajax的缺点和在IE下的问题？

相关阅读：<http://www.cnblogs.com/SanMaoSpace/archive/2013/06/15/3137180.html>

1. HTTP状态码

相关阅读：<http://http.phpddt.com/>

1. XML和JSON的区别？

相关阅读：<http://www.cnblogs.com/SanMaoSpace/p/3139186.html>

1. 如何解决跨域问题？
2. Jsonp
3. iframe跨域
4. script元素的src属性处理
5. 在服务器使用get处理
6. flash跨域

相关阅读：<http://www.jb51.net/article/58156.htm>

1. 请描述一下 cookies，sessionStorage 和 localStorage 的区别？

**共同点：**都是保存在浏览器端，且同源的。

**区别：**

1. cookie数据始终在同源的http请求中携带（即使不需要），即cookie在浏览器和服务器间来回传递；cookie数据还有路径（path）的概念，可以限制cookie只属于某个路径下。存储大小限制也不同，cookie数据不能超过4k，同时因为每次http请求都会携带cookie，所以cookie只适合保存很小的数据，如会话标识。
2. 而sessionStorage和localStorage不会自动把数据发给服务器，仅在本地保存。sessionStorage和localStorage 虽然也有存储大小的限制，但比cookie大得多，可以达到5M或更大。
3. 数据有效期不同，sessionStorage：仅在当前浏览器窗口关闭前有效，自然也就不可能持久保持；localStorage：始终有效，窗口或浏览器关闭也一直保存，因此用作持久数据；cookie只在设置的cookie过期时间之前一直有效，即使窗口或浏览器关闭。
4. 作用域不同，sessionStorage不在不同的浏览器窗口中共享，即使是同一个页面；localStorage 在所有同源窗口中都是共享的；cookie也是在所有同源窗口中都是共享的。Web Storage 支持事件通知机制，可以将数据更新的通知发送给监听者。Web Storage 的 api 接口使用更方便。
5. 如何删除一个cookie？
6. 应用setcookie()函数删除cookie  
   删除Cookie和创建Cookie的方式基本类似,删除Cookie也使用setcookie()函数,删除Cookie只需要将setcookie()函数中的第二个参数设置为空值,第三个参数将Cookie的过期时间设置为小于系统的当前时间即可.  
   2.应用浏览器手动删除Cookie  
   在使用Cookie时,Cookie自动生成一个文本文件存储在IE浏览器的Cookie临时文件夹中,应用浏览器删除Cooki文件
7. 什么是web worker，及如何使用？

**定义：**

Web Worker 可以让你在后台运行Javascript

**简介：**

一般来说Javascript和页面的UI会共用一个线程，所以当点击一个按钮开始运行Javascript后，在这段代码运行完毕之前，页面是无法响应用户操作的，换句话来说就是被“冻结”了。而这段代码可以交给Web Worker在后台运行，那么页面在Javascript运行期间依然可以响应用户操作。后台会启动一个worker线程来执行这段代码，用户可以创建多个worker线程。所以你可以在前台做一些小规模分布式计算之类的工作，不过Web Worker有以下一些使用限制：

1. Web Worker无法访问DOM节点；
2. Web Worker无法访问全局变量或是全局函数；
3. Web Worker无法调用alert()或者confirm之类的函数；
4. Web Worker无法访问window、document之类的浏览器全局变量；

不过Web Worker中的Javascript依然可以使用setTimeout(),setInterval()之类的函数，也可以使用XMLHttpRequest对象来做Ajax通信。

相关阅读：<http://blog.csdn.net/dojotoolkit/article/details/25030289>

1. Audio和video对象常用的属性和方法有哪些？

**<audio>标签属性：**

1. src：音乐的URL
2. preload：预加载
3. autoplay：自动播放
4. loop：循环播放
5. controls：浏览器自带的控制条

**Html代码**

<audio id="media" src="http://www.abc.com/test.mp3" controls></audio>

**<video>标签属性：**

src：视频的URL

1. poster：视频封面，没有播放时显示的图片
2. preload：预加载
3. autoplay：自动播放
4. loop：循环播放
5. controls：浏览器自带的控制条
6. width：视频宽度
7. height：视频高度

**Html代码**

<video id="media" src="http://www.abc.com/test.mp4" controls width="400px" heigt="400px"></video>

1. 获取HTMLVideoElement和HTMLAudioElement对象

**Js代码**

//audio可以直接通过new创建对象

Media = new Audio("http://www.abc.com/test.mp3");

//audio和video都可以通过标签获取对象

Media = document.getElementById("media");

**Media方法和属性：**

HTMLVideoElement 和HTMLAudioElement 均继承自HTMLMediaElement

**Js代码**

**//错误状态**

Media.error; //null:正常

Media.error.code; //1.用户终止 2.网络错误 3.解码错误 4.URL无效

**//网络状态**

Media.currentSrc; //返回当前资源的URL

Media.src = value; //返回或设置当前资源的URL

Media.canPlayType(type); //是否能播放某种格式的资源

Media.networkState; //0.此元素未初始化  1.正常但没有使用网络  2.正在下载数据  3.没有找到资源

Media.load(); //重新加载src指定的资源

Media.buffered; //返回已缓冲区域，TimeRanges

Media.preload; //none:不预载 metadata:预载资源信息 auto:

**//准备状态**

Media.readyState;    //1:HAVE\_NOTHING 2:HAVE\_METADATA 3.HAVE\_CURRENT\_DATA 4.HAVE\_FUTURE\_DATA 5.HAVE\_ENOUGH\_DATA

Media.seeking; //是否正在seeking

**//回放状态**

Media.currentTime = value; //当前播放的位置，赋值可改变位置

Media.startTime; //一般为0，如果为流媒体或者不从0开始的资源，则不为0

Media.duration; //当前资源长度 流返回无限

Media.paused; //是否暂停

Media.defaultPlaybackRate = value;//默认的回放速度，可以设置

Media.playbackRate = value;//当前播放速度，设置后马上改变

Media.played; //返回已经播放的区域，TimeRanges，关于此对象见下文

Media.seekable; //返回可以seek的区域 TimeRanges

Media.ended; //是否结束

Media.autoPlay;  //是否自动播放

Media.loop;  //是否循环播放

Media.play();    //播放

Media.pause();   //暂停

**//控制**

Media.controls;//是否有默认控制条

Media.volume = value; //音量

//TimeRanges(区域)对象

TimeRanges.length; //区域段数

TimeRanges.start(index) //第index段区域的开始位置

TimeRanges.end(index) //第index段区域的结束位置

**事件：**

Js代码

eventTester = function(e){

Media.addEventListener(e,function(){

 console.log((new Date()).getTime(),e);

});

}

eventTester("loadstart");   //客户端开始请求数据

eventTester("progress");    //客户端正在请求数据

eventTester("suspend");     //延迟下载

eventTester("abort");       //客户端主动终止下载（不是因为错误引起），

eventTester("error");       //请求数据时遇到错误

eventTester("stalled");     //网速失速

eventTester("play");        //play()和autoplay开始播放时触发

eventTester("pause");       //pause()触发

eventTester("loadedmetadata");  //成功获取资源长度

eventTester("loadeddata");  //

eventTester("waiting");     //等待数据，并非错误

eventTester("playing");     //开始回放

eventTester("canplay");     //可以播放，但中途可能因为加载而暂停

eventTester("canplaythrough"); //可以播放，歌曲全部加载完毕

eventTester("seeking");     //寻找中

eventTester("seeked");      //寻找完毕

eventTester("timeupdate");  //播放时间改变

eventTester("ended");       //播放结束

eventTester("ratechange");  //播放速率改变

eventTester("durationchange");  //资源长度改变

eventTester("volumechange");    //音量改变

1. 如何配置离线存储？
2. 首先来讲解下离线存储的使用方法，说起来也很简单。只要在你的页面头部像下面一样加入一个manifest的属性就可以了。

<!DOCTYPE HTML><html manifest = "cache.manifest">......</html>

1. 然后cache.manifest文件的书写方式，就像下面这样：

CACHE MANIFEST

#v0.11

CACHE:

js/app.js

css/style.css

NETWORK:

resourse/logo.png

FALLBACK:/ /offline.html

离线存储的manifest一般由三个部分组成:

1. CACHE:表示需要离线存储的资源列表，由于包含manifest文件的页面将被自动离线存储，所以不需要把页面自身也列出来。
2. NETWORK:表示在它下面列出来的资源只有在在线的情况下才能访问，他们不会被离线存储，所以在离线情况下无法使用这些资源。不过，如果在CACHE和NETWORK中有一个相同的资源，那么这个资源还是会被离线存储，也就是说CACHE的优先级更高。
3. FALLBACK:表示如果访问第一个资源失败，那么就使用第二个资源来替换他，比如上面这个文件表示的就是如果访问根目录下任何一个资源失败了，那么就去访问offline.html。

相关阅读：<http://www.cnblogs.com/shoestrong/p/6435169.html>

1. 什么是事件委托？

使用事件委托技术能让你避免对特定的每个节点添加事件监听器；相反，事件监听器是被添加到它们的父元素上。事件监听器会分析从子元素冒泡上来的事件，找到是哪个子元素的事件。

相关阅读：<http://www.webhek.com/post/event-delegate.html>

1. ES6有哪些新增特性？
2. 迭代器
3. 箭头函数
4. 类的支持
5. 增强的对象字面量
6. 字符串模板
7. 解构
8. 参数默认值，不定参数，拓展参数
9. let与const 关键字
10. for of 值遍历
11. Proxies 监听对象身上发生了什么事情
12. Symbols 一种基本类型
13. Promises 处理异步操作的一种模式
14. Math，Number，String，Object 的新API