**1.语句和表达式**

语句相当于句子，表达式相当于短语，运算符相当于标点符号和连接词。

js中的表达式可以返回一个结果值。默认情况下，控制台会显示所执行的最后一条语句的结果。

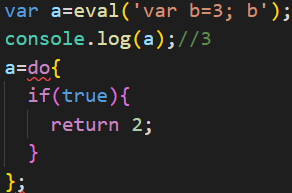
var a;//声明语句

b=a;//赋值表达式

b;//表达式语句

如果在控制台输入var a=42;得到的结果是undefined而不是42。因为这个语句没有返回值。

语法不允许我们将语句的结果赋值给一个变量，但可以用eval，另外es7有个do提案。



++a++报错

前面的先执行，++a变成一个数字，所以数字++报错

clipboard.png

delete

delete用来删除对象中的变量，成功返回true，否则返回false

delete不能删除全局变量，删除不掉

delete删除不了原型链中的变量

delete删除数组中的元素后，数组长度不变，

用逗号,连接一系列语句时，它的优先级最低，从左到右执行语句

有时js会自动为代码行补上缺失的分号，即分号自动插入(ASI)，只在换行时插入。ASI实际上是一个纠错(解析器错误)机制。

按照ECMAScript标准，一些特定语句必须以分号结尾。有时候，这些分号被省略了，解释器会自己判断语句该在哪里终止，这种行为叫自动插入分号，简称ASI。实际上分号被没有被插入。这些特定的语句有：空语句 let const important export 变量赋值 表达式 debugger continue break return throw

ASI包括三条规则和2种例外，

三条规则描述何时该自动插入分号



restricted production表示有严格限定的语法

两种例外表示如果被解析成下面的样子，就不插入分号

clipboard.png

js有运行时错误(TypeError ReferenceError SyntaxError等)，还有编译时错误(语法错误等)。程序有语法错误会无法运行

**2.对象和原型**

对象是通过函数创建的，函数又是一种对象

对象实例的\_\_proto\_\_等于对象的prototype

每个函数都有一个prototype，每个对象都有一个\_\_proto\_\_指向创建该对象的函数的prototype

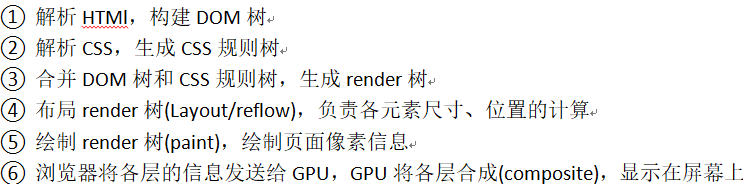
Object.prototype的\_\_proto\_\_指向null

a instanceof A

沿着a的\_\_proto\_\_这条线找，同时沿着A的prototype这条线找，如果能找到同一个引用则返回true，\_\_proto\_\_这条线即原型链。

Object.prototype.hasOwnProperty()自身是否含有某属性

**3.浏览器渲染过程**



前3点可能要多次执行，比如js脚本操作DOM、更改CSS样式时，浏览器要重新构建DOM、CSSOM树，要重新render，重新layout、paint

应该将css放在头部，将js放在尾部。构建render树需要DOM和CSSOM，所以要让css尽早加载，缩短首次渲染时间。js引擎和渲染引擎是互斥的。

首屏优化的方法：减少资源的请求数量、使CSS样式表尽早加载，不使用@important而使用link，异步js(async)

**4.回流和重绘**

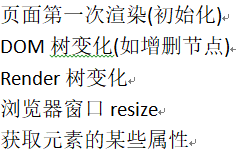
reflow回流：元素的内容、结构、位置或尺寸发生了变化

repaint重绘：元素发生的改变只影响了节点的一些样式，如背景色、边框颜色，文字颜色等

回流的成本开销高于重绘，一个节点的回流往往导致子节点以及同级节点的回流

回流一定重绘，重绘不一定回流

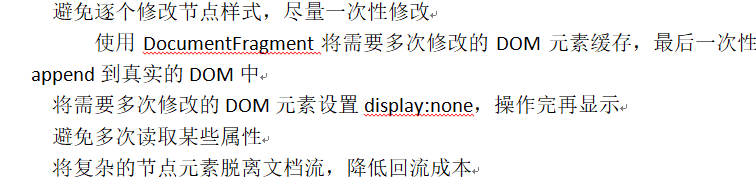
引起回流：



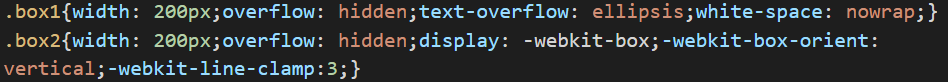
引起重绘：

背景色、颜色、字体改变，字体大小发生变化时，会触发重绘

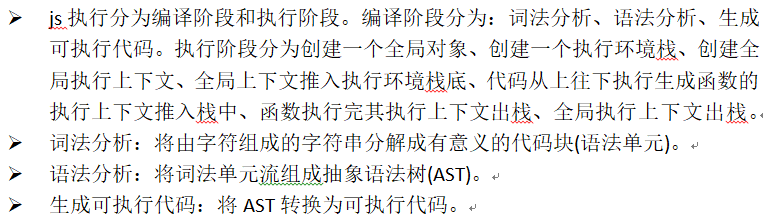
避免回流/重绘



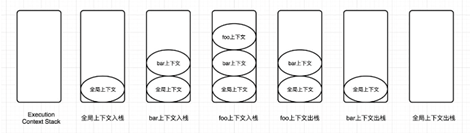
**5.单行文本溢出/多行**



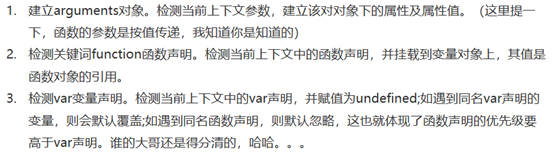
**6.js执行**



执行环境栈：



创建变量对象：



变量对象(Variable Object)和活动对象(Active Object)的区别

他们其实都是同一个对象，只是处于执行上下文的不同生命周期。在执行阶段变量对象变为活动对象。不过只有处于函数调用栈栈顶的执行上下文中的变量对象，才会变成活动对象。

**7.内存泄漏**

不再用到的内存没有及时释放，叫内存泄漏

垃圾回收机制即内存自动管理，常用的方法是引用计数

内存泄漏的识别方法：如果连续5次垃圾回收之后，内存占用一次比一次大，就有内存泄漏

WeakMap WeakSet 中的对象是弱引用 垃圾回收机制不考虑WeakSet对该对象的引用，即如果其他对象不再引用该对象，那么垃圾回收机制会自动回收该对象占用的内存。

**8.toString()**

null和undefined没有toString()

布尔值加引号返回

字符串原值返回

数值类型：

正浮点数、NaN、Infinity加引号返回

负浮点数及加'+'的浮点数调用toString()，相当于先执行toString()再添加正负号变成数字

整数调用toString()会报错，因为点会被识别为小数点，因此要用小括号包围起来

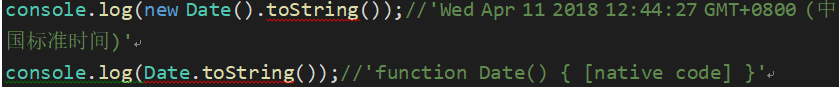
toString()可以添加第二个参数，表示进制

对象及自定义对象返回'[object Object]'

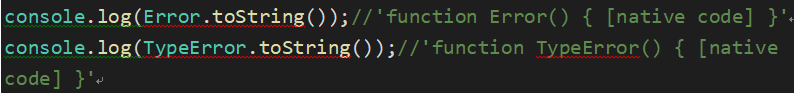
函数返回函数具体内容

数组类型变成以逗号分隔的字符串(空数组返回空字符串)

日期类型：



Error类型



**9.valueOf()**

布尔值：

clipboard.png

字符串：

clipboard.png

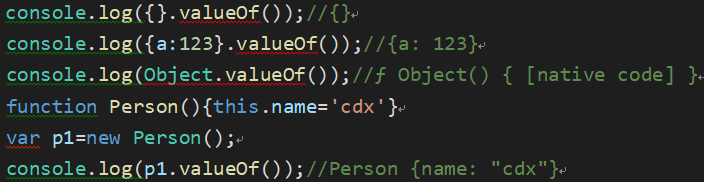
数字：

整数必须加小括号，返回原值

浮点数、正负数也返回原值

-0的valueOf()是-0 -0的toString()是0

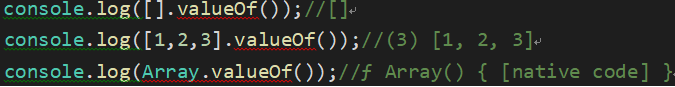
对象：



函数：

clipboard.png

数组：



日期：

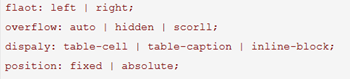
clipboard.png

Error：

clipboard.png

**10.BFC块级上下文**

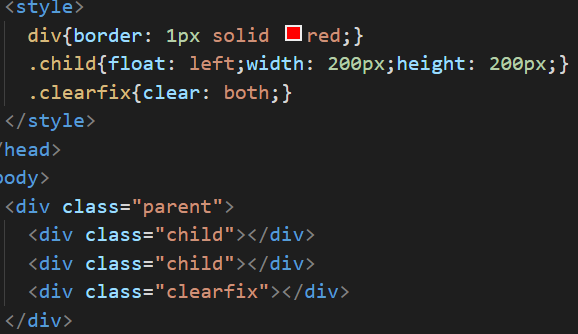
触发BFC的条件：



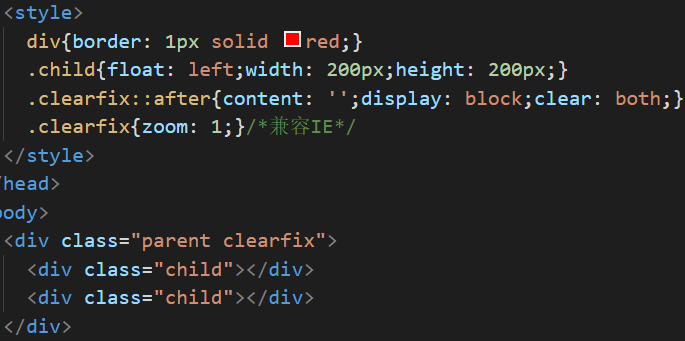
**11.清除浮动**

子元素设置浮动会导致父元素高度塌陷

方法1：给父元素添加一个空的字元素，为其设置clear:both;



方法2：使用after伪元素



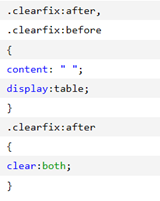
方法3：给父元素设置高度

方法4：给父元素设置overflow:hidden;

方法5：给父元素设置浮动

方法6：给父元素设置绝对定位

常用：



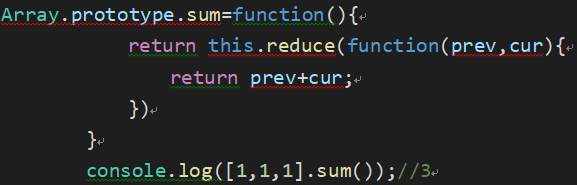
**12.reduce**

reduce方法接收一个函数作为累加器，第二个参数(可选)为初始值

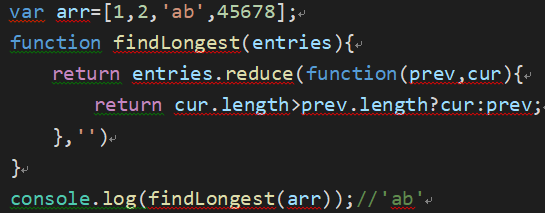
此回调函数有4个参数：上一次调用返回值或初始值、当前值、当前索引、arr数组

reduce方法的返回类型和初始值相同

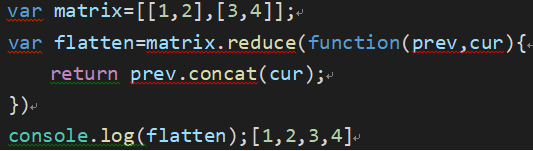
求和：



由于reduce方法依次处理每个元素，所以可用来搜索某个元素，比如找出长度最长的数组元素



扁平化二维数组：



在空数组上，不带初始参数调用reduce会导致类型错误异常。

**13.map**

map对数组的每一项运行给定函数，返回每次函数调用的结果组成的数组

map中函数的参数：当前元素、当前索引、当前数组

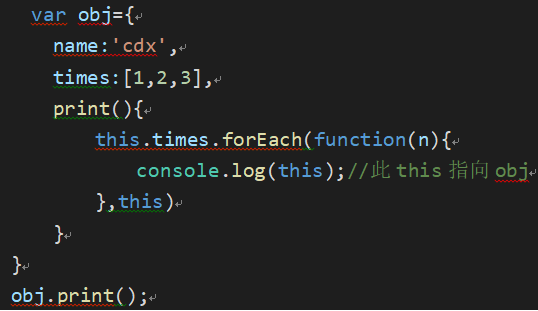
map还可以接收第2个参数，表示回调函数执行时this的指向

**14.forEach**

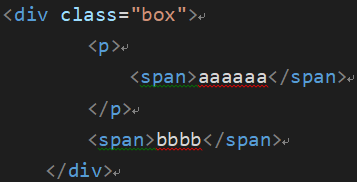
forEach方法对数组中的每一项运行给定函数，没有返回值

forEach还可以接收第2个参数，表示回调函数执行时this的指向

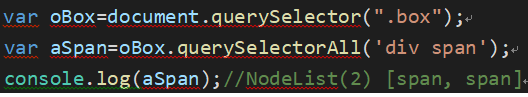
多层循环的this指向是不确定的，可以使用forEach的第二个参数固定this



**15.querySelector/querySelectorAll**



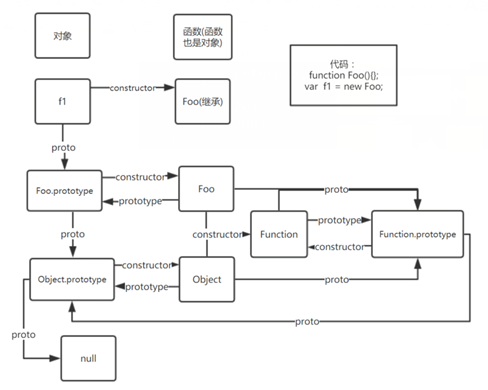
缺陷：



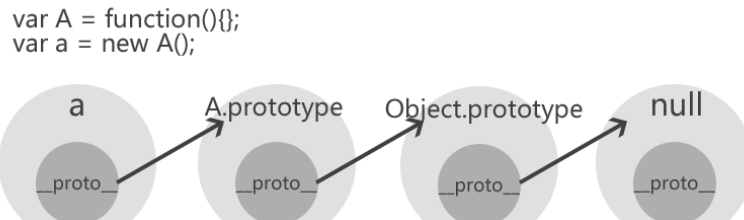
aSpan应该是oBox里面的祖先为div的span元素，这个div不应该包括oBox本身。解决办法是在selector前面指定调用元素的id限制匹配范围



**16.constructor prototype proto**



prototype是函数的属性，\_\_proto\_\_是对象的属性



\_\_proto\_\_链即原型链，当查找对象属性时，首先在对象进行查找，然后沿原型链向上查找