Js 高级程序设计

第一章

第二章

1. 异步脚本
   * 1. Defer
     2. async
2. 文档模式
3. 渲染时间线

第三章

1. 变量区分大小写
2. 变量的命名规则：字母、下划线（\_）、美元符（$） 开头， 建议驼峰式命名
3. 严格模式 ES5提出了严格模式，旨在解决js语言中的一些不规范行为
   1. 变量使用前必须申明
4. 数据类型：
   1. Number
   2. Boolean
   3. String
   4. Object
   5. Null
   6. Undefined
5. Number类型
   1. 使用IEEE754表示整数和浮点数
   2. 基本表示是10进制，也可以用8进制（0），16进制（0x）
   3. 浮点数保存的空间是整数的两倍，ECMAScript会不失时机的将浮点数转换成整数
   4. 浮点数最高精度为17位，计算时可能有问题 如：0.1 + 0.2 == 0.30000000000000004. 这是基于IEEE754数值的问题
   5. NaN 不是数字的数值 (not a number)
   6. isNaN 函数，判断是否是NaN, 这个函数有个问题，如果传入的参数不是数字类型，会将其转换成数字类型，会调用Number（）方法，参数也可以是对象，这就涉及到对象转换成原始值的问题了，具体看后面的
   7. Number(params)
      1. true -> 1; false -> 0
      2. null -> 0;
      3. undefined -> NaN
      4. 字符情况 （忽略开头和结尾的空格、换行符）
         1. 字符数字，忽略前导0，转换成对应的10进制数字， ‘011’ -> 11
         2. 字符16进制数字，转换成10进制数字 ‘0xf’ -> 15
         3. 字符串为空或者全为空格(换行符等都行)，转换成0
         4. 其他情况为NaN
      5. 对象情况，先转换成原始值（value, toString），再进行上述规则
   8. parseInt(string, ?radix) radix：基底(2-36)，表示传入的字符串的进制,默认为10。返回的结果是10进制（前提能够转换成数字）
      1. 传入的参数不是字符串，会将参数转换成字符串
      2. 字符串情况
         1. 忽略字符串前面的空格，直到找到第一个非空字符。如果不是数字或者 + - 则直接返回NaN。继续识别字符，直到不能转换成数字为止
            1. parseInt(‘sfdsfdsf12’) -> 12
            2. parseInt(‘021’) -> 21
            3. parseInt(‘021’, 8) -> 17
            4. parseInt(true) -> NaN true 转换成字符串 ‘true’
            5. parseInt(true, 30) -> 897 ‘t’ 在30进制中表示29，’r’ 表示27 即 29 \* 30 + 27 = 897
6. String
   1. 字符串创建之后是不变的，要改变某个变量保存的字符串，首先要销毁原来的字符串，然后再用另一个包含新值的字符串填充该变量
   2. toString(?radix)
      1. 可选基数，转换成radix 进制的字符串,很少用到
      2. null undefined 没有toString 方法（没有原型）
      3. String方法细节：
         1. 如果参数有toString 方法，调用toString 方法,返回对应结果
         2. 对象如果重写了toString方法，注意，如果返回的值不是原始值，会报错
         3. null undefined --> ‘null’ ‘undefined’
7. 位运算符: 位运算在底层是以32位进行计算的, ECMA中整数有两个类型，有符号整数和无符号整数（只允许整数），有符号整数用31位表示数值，第32位表示符号，0为正，1为负。正数以真2进制新式存储，负数为对应正数的补码
   1. 按位非（~）返回数字的反码
   2. 按位与（&）将两个数位对齐，如果为1返回1，否则0
   3. 按位或（|）和上面类型，只要有一个为1返回1，否则0
   4. 按位异或（^）两个数，如果对应位数字不同返回1，否则0
   5. 左移（<<） 将所有位（符号位除外）向左移动,右侧多出的位用0补充，
   6. 有符号右移（>>）所有位（符号位除外）向右移动，多出的用符号位来填充
   7. 无符号右移（>>>）所有位向右移动，符号位也动，多出的用0填充
8. 布尔运算符
   1. 非（！）一个操作数，首先转换成布尔值，再取反
      1. 调用Boolean方法转换成布尔值，该放下转成成false的值为0、false、’’、NaN、null、undefined 。其他为true
   2. 与（&&）两个操作数 a, b
      1. 如果a转换成布尔值为true 返回 b的值
      2. 如果a转换成布尔值为false 返回 a 的值
      3. 如果b有错误，a转换成布尔值为false, 不会报错
   3. 或（||）两个操作数a,b
      1. 如果a转换成布尔值为true, 返回a的值
      2. 如果a转换布尔值为false, 返回b的值
9. 类型转换
10. 逗号运算符：返回最后一个表达式的值
    * + 1. 变量的复制
           1. 原始值：复制值
           2. 引用值：复制引用（指针）
        2. 参数的传递也和变量复制一样
        3. 类型判断
           1. typeof: 判断数据的类型，返回的值时字符串

typeof(null) --> ‘object’

typeof({}) --> ‘object’

typeof(function(){}) --> ‘function’

typeof(1) --> ‘number’

typeof(true) --> ‘boolean’

typeof(undefined) --> ‘undefined’

注意：null 返回 object, 而且 给定一个未定义的变量(var 情况下)，使用typeof 并不会报错。返回值时string类型，所以连续两次调用返回’string’

* + - * 1. Instanceof: A instanceof B: A的原型链上是否有B的原型

A：是一个引用值. B 是函数

Object.prototype 一般是原型的顶端。 所以 [] instanceof Object --> true; function () {} instanceof Object ---> true; 这是由于 [].\_\_proto\_\_.proto\_\_ === Object.prototype. 同理 函数也类似

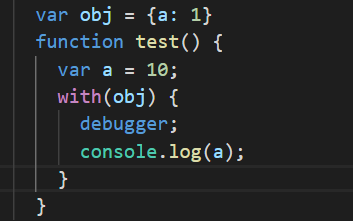
Array.prototype.\_\_proto === Object.prototype

Function.prototype.\_\_proto === Object.prototype

Object.\_\_proto\_\_ === Function.prototype

A 为原始值将返回 false

* + - 1. 执行环境和作用域
         1. 执行环境定义了变量或函数有权访问其他的数据，决定了他们的访问规则。每个执行环境都有一个相对应的变量对象，里面存储着变量和函数
         2. 全局执行环境是最外层的一个执行环境，根据ECMAscript实现所在的宿主环境不同，表示的执行环境也不同。在Web浏览器中，全局执行环境被认为是window对象（还是有区别）。
         3. 某个执行环境的所有代码执行完之后，改环境会被销毁（其中的变量与函数都被销毁）。全局执行环境在应用退出时关闭（在浏览器中，为关闭网页）
         4. 每个函数都有执行环境。当执行流进入一个函数时，会将函数环境压入环境栈中，执行完后，在栈中弹出。
         5. 当代码在一个执行环境中执行时，会创建一个作用域链。其用途是保证对执行环境有权访问的所有变量和函数的有序访问
         6. 作用域的前端始终是当前执行代码所在的环境的变量对象。如果这个环境是函数，则将其活动对象（activity object）作为变量对象。作用域链中的下一个变量对象来自外部环境。直到全局环境
         7. 在环境中访问变量或者函数，会从作用域链的前端开始，逐级搜索，直到找到， 或者找不到为止（通常会报错）
      2. 作用域延长
         1. 执行环境分为全局和函数，但是也有方法延长作用域链。有些语句在执行时，可以在作用域的前端临时加入一个变量对象，该对象会在代码执行完之后销毁
         2. with

with语句会将制定的对象添加到作用域中如： with语句执行时会将obj对象添加到作用域的前端，所以此时a的值是1而非10，注意一点，在with语句中申明的变量还是会挂在最近的函数或者全局作用域中

* + - * 1. try-catch的catch 块
      1. 垃圾回收机制
         1. 标记清除
         2. 引用计数

记录每个值被引用的次数

问题，循环引用

* + - 1. 数组检测
         1. Instanceof的局限，在网页嵌入了多个框架的时候，由于有多个全局环境，从而存在不同的Array构造函数
      2. 数组方法
         1. pop 移除数组最后一个元素并返回
         2. push 将元素推入数组的末尾，返回长度
         3. unshift 将元素推入数组的首位，返回长度
         4. shift 移除数组的第一个元素并返回
         5. reverse 反转数组
         6. sort 默认升序排列，而且会数组每一项转换成字符串，然后进行排序，即使每一项是数字，比较的也是字符串的顺序。可以接受一个比较函数，该函数接收2个参数，如果第一个参数在第二个参数前面，则返回负数，之后返回正数
         7. splice 功能很强大

添加：3个参数及以上，第一个是起始位置，第二个为0，后面的为插入的项

删除：2个参数，第一个起始位置，第二个为删除的个数

替换: 第一个起始位置，第二个为删除的个数，后面为替换的项

该方法返回删除的元素组成的数组，可能为空数组

* + - * 1. concat 基于数组创建一个新数组，然后将副本拷贝进去，将参数添加到新数组的末尾，如果参数是数组，会进行一层扁平化。
        2. slice（start, end） 基于原数组创建一个新数组

接收2个参数，分别是开始位置、结束位置（不含）。返回原数组开始位置到结束位置的项组成的新数组。

不传参数，则拷贝整个数组。

传一个参数，则返回该参数为起点，到数组的最后一项（含）组成的数组

如果参数为负，会加上数组的长度。如果加上数组的长度还小于0则会将改参数置为0

第二个参数如果小于等于第一个参数 则返回空数组

* + - * 1. indexOf（item, ?index） item: 要查找的项。 index: 开始索引，默认为0。如果找到返回该位置的索引，没有返回-1
        2. lastIndexOf（item, ?index）和indexOf 类似，区别在于是从数组末尾开始查找
        3. every 数组每一项都运行给定函数，如果每一项都返回true,则返回true。如果返回false就提前结束
        4. ome 数组每一项都运行给定函数，如果有项都返回true,则返回true（结束）。如果全为false,则返回false
        5. filter 数组每一项运行给定函数，返回函数返回值为true的项组成的数组
        6. map 数组每一项运行给定函数，返回函数返回值组成的数组
        7. forEach 数组每一项运行给定函数，没有返回值（不能使用break, continue）
        8. reduce 从数组的第一项开始，遍历到最后一项，

有两个参数，参数函数4个参数，前一个值(prev)，当前值(cur)，当前索引(index)，数组对象(arr)。如果没有初始值，就将数组的第一个项作为prev，每次执行完的结果作为下次的prev

可选的初始值

* + - * 1. reduceRight 和 reduce类似，从数组最后一项开始
      1. 正则表达式
         1. 模式

g 全局匹配

i 忽略大小写

m 多行匹配

* + - * 1. 实例方法

exec(str) 接收一个字符串，返回第一个与str匹配项的数组，没有匹配到返回null

返回的数组是Array的实例，但是额外有index, input属性。Index属性指的是匹配项在字符串中的位置，input表示应用正则表达式的字符串。

数组的第一项是与正则表达式匹配的字符串，其他项与模式中捕获组匹配的项（如果有）

在设置了g标志时，exec每次也只会返回一个与之匹配的项。每次匹配的时候，会从上次匹配结束点继续匹配（lastIndex记录上次的位置）

如果没有g标志，则每次都从头开始匹配

test(str) 如果匹配成功返回true, 反之false

* + - 1. 函数
         1. 没有重载
         2. 函数申明、函数表达式
         3. arguments、arguments.callee、arguments.caller
         4. call: 改变this 指向， 参数一个个传递
         5. apply 改变this指向，参数按照数组传递
         6. bind 改变this指向
      2. 包装类
         1. String
         2. Number
         3. Boolean
         4. 在访问字符串的属性时，会将字符串转换成对应的包装类对象，然后在访问，并在执行完之后，销毁这个对象、
      3. 字符串
         1. 方法

charAt

charCodeAt

concat

substr

slice

substring

indexOf

lastIndexOf

trim

replace

match

search

* + - 1. Global对象
         1. 全局方法

URI编码方法

encodeURI：不会对本身属于 URI 的特殊字符进行编码，例如冒号、正斜杠、问号和井字号

encodeURIComponent：对它发现的任何非标准字符进行编码

eval

接受一个参数，需要执行的代码或者字符串。

eval是魔鬼zzz。

当解析器发现代码中调用 eval()方法时，它会将传入的参数当作实际的 ECMAScript 语句来解析，然后把执行结果插入到原位置。通过 eval()执行的代码被认为是包含该次调用的执行环境的一部分，因此被执行的代码具有与该执行环境相同的作用域链。这意味着通过 eval()执行的代码可以引用在包含环境中定义的变量

在 eval()中创建的任何变量或函数都不会被提升，因为在解析代码的时候，它们被包含在一个字符串中；它们只在 eval()执行的时候创建

严格模式下，在外部无法访问到eval内部申明的变量或者函数

1. 面向对象的程序设计
   * + 1. 对象：无序属性的集合，属性可以包括基本值、引用值
          1. 属性类型: ES5定义了只有内部才能用的特性，描述了属性的各种行为

[[Configurable]]: 表示能否配置属性。如删除属性、修改属性的特性

[[Enumerable]]：表示能够通过for-in 循环返回属性

[[Writable]]：表示属性是否可写

[[Value]]： 表示属性的值

[[Get]]： 读取属性时调用的函数

[[Set]]：设置属性时调用的函数

以上的属性只能通过Object.defineProperty去设置，如果没有指定，则为false。通过对象字面量的方式创建，默认为true(configurable、enumerable、writable)

属性的value值 与 get/set 不能同时存在

Object.defineProperties 定义多个属性

Object.getOwnPropertyDescriptor 获取属性的特性值

* + - 1. 创建对象
         1. 工厂模式
         2. 构造函数模式
         3. 原型模式
         4. 组合

1. 函数
   * + 1. 作用域
       2. 闭包
       3. This指向
       4. Call、apply、bind
       5. 私有化变量