# Js基础复习汇总

* 提到js就有三个东西：ecmascript DOM BOM
* 解释型语言 ： 不需要编译
* 基本数据类型
* 六种数据类型：
  + String
  + Number
  + Boolean
  + NULL
  + Undefined
  + Obeject
* 在字符串中我们可以使用\作为转义字符
* 在js中所有数值都是number类型，包括小数
* Typeof检查一个值的类型
* Js中表示数字的最大值 ： Number.MAX\_VALUE
  + 如果NUMBER表示的数字超过了最大值，就会显示infinity（无穷）
  + Infinity是数字类型，用typeof可得知
  + NaN是一个特殊的数字表示：不是一个数字
* 使用typeof检查Null时，会显示object类型，因为Null其实就是空对象
* Undefined是声明后，没有赋值时才会出现的类型
* 数据转换
* 转为string
  + 调用被转换数据的Tostring（）方法
    - Null和undefined是不能tostring的，会报错
    - 数字调用tostring()可以传参数,指定数字转成其他进制
  + 调用Stiring（）函数
    - Null和undefined也是可以转 的，Null 🡪“Null” undefined 🡪” undefined”
  + 任何数据类型 + “”
* 转为number
  + 使用Number()函数
    - 只要有不是数字的内容时，会返回NaN
    - 如果字符串是空或者全是空格，则会返回0
    - 如果是布尔值 ，会转为1（true） 和 0（false）
    - Null转数字，会返回0
    - Undefined转数字，会返回NaN
  + ParseInt() ，ParseFloat()
    - ParseInt()此方法专门用来对付字符串，可以将字符串中的有效的整数内容取出来然后转换成Number 如 ： “123px” 🡪 123
    - ParseFloat作用于ParseInt类似，但是他可以取出小数
    - 此函数的参数如果传的是非string，会先将此参数转成string再去操作
    - parseInt(a,10)，可以将“070”等0开头的8进制的数字字符串，转换为10进制，默认是8进制转为8进制
* 转为布尔
  + Boolean()
    - 数字转布尔 除了0和NaN，其他都是true
    - 字符串转布尔，除了空字符，其他都是True
    - Null 和undefined都会转换为false
    - 对象会转换为True
* 运算符
* Typeof
* 当对非number类型进行运算时，会将其先转成number再进行运算，除了两个字符串进行加法运算（此情况会拼接）
* 任何值和NaN做运算，都得NaN
* 任何值和字符串做加法运算，都会将其转为字符串然后再拼接，可以做隐式字符串转换
* 可以用一元运算符的+ ,对字符串进行隐式数字转换 如： +“123” 🡪 123
* 关系运算符，比较两个字符串时，时比较字符编码，而且是一位一位进行比较的
* 在js字符串中使用转义字符“\u+unicode编码“来输出
* 在hmtl中使用转义字符 #& +10进制的unicode编码
* 相等运算：
  + ==：
    - 类型不同，值相同时，会转换为相同类型再比较
    - Undefined 和null做==判断时，返回true
    - NaN不和任何值相等，包括他本身 （需要判断一个值是否是NaN用函数isNaN）
  + ===:
    - 不会做自动类型转换，类型不同，值相同也直接返回false
* 三元运算符 ： 条件式 ？语句 ： 语句
* 语句注意点
* Js中的代码块只具有分组作用，其他无作用
* Break只能用来退出循环或者switch，不能在if中单独使用
* Break只对离最近的循环产生影响(结束当前循环)
* 可以在循环前建一个label来识别当前循环 如 ： label : if(){}
  + 使用break 配合label可以结束lable所在的循环
  + 可以在内循环中Break label来结束外循环
* **Js中对象**
* 对象类型，创建对象
  + 内建对象：由ES标准中定义的对象，在任何的ES的实现中都可以使用
    - Math,String之类的
  + 宿主对象
    - Dom bom之类的浏览器提供的对象
  + 自定义对象
    - 由开发人员自己创建的对象
  + 创建对象：
    - New 关键字调用
    - Var obj = {} :大括号就是对象字面量
* 属性类型，创建属性
  + 添加/修改属性
    - 对象.属性 = “”
    - 对象【“属性”】= “”
  + 读取属性
    - 对象.属性
    - 对象【“属性”】= “”
    - 如果读取对象中没有的属性，会返回undefined，不会报错。
  + 删除对象
    - Delete 对象.属性
  + 在js对象的属性值，可以是任意的数据类型
    - 甚至也可以是个对象
  + 检查对象中是否含有属性
    - Console.log(属性 in 对象) 有则返回true
  + Js中的变量（基本数据类型）是保存到栈内存中，对象（引用数据类型）是保存在堆内存中
    - 比较基本数据类型的值时，就是比较值
    - 比较引用数据类型时，是比较其引用，所以两个长得完全一样的对象用==时，会得出false
* Js中的函数
* 函数也是一个对象
* 创建函数：
  + Var 函数名 =New Function()
    - 此时通过new出Function构造函数的实例来创建普通函数
  + function 函数名([形参1，形参2，…]){}
  + Var 函数名 = function([形参1，形参2，…]){}
* 函数的实参，可以是任何值，包括对象或者函数
* 函数的返回值，可以是任何值，包括对象或者函数
* 函数对象可以立即调用：
  + （function(a,b){。。。。。}）(“123”,”234”)
* 函数也可以作为对象的属性，此时调用对象的函数可以说是对象的方法（名称上的区别）
* 对象的枚举：
  + For (var n in 对象 ){}
  + For…in语句中，对象有几个属性，就循环几次
  + 在循环中获取属性的值是需要用obj[属性]的方式获取值
* Js中的作用域
* 全局作用域
  + 直接编写在script标签中的js代码 ，都在全局作用域
  + 全局作用域在页面打开时创建，在页面关闭时销毁
  + 全局作用域有一个全局对象window。
    - 它代表浏览器window，它由浏览器创建，我们可以直接使用
    - 在全局作用域中，创建的变量都会作为window对象的属性保存
    - 在全局作用域中，创建的函数都会作为window对象的方法保存
  + 变量的声明提前
    - 使用var关键字声明或者赋值的变量，会在所有的代码执行之前被声明（只是声明，不赋值）
  + 函数的声明提前
    - 使用函数声明形式创建的函数function 函数名（）{} 会在所有代码执行之前被声明且创建
    - 如果使用var关键字声明函数的话，var f = function(){} ，不会再所有代码执行前创建，只会声明var f；（跟变量的声明提前一样）
* 函数作用域
  + 调用函数时创建函数作用域，函数执行完毕后，函数作用域销毁
  + 每调用一次函数就会创建一个新的函数作用域，他们之间是互相独立的
  + 在函数作用域中可以调用全局作用域的变量，
    - 但是全局中访问不了函数作用域的变量
  + 在函数作用域操作一个变量时，会先在在自身作用域中寻找，有的话就直接使用，没有的话就上一级作用于中查找，一直找到全局作用域，如果还是找不到就报错
  + 在函数作用域中想要直接访问全局的变量时，可以直接用window.变量
  + 变量声明提前（与全局作用域一致）
  + 函数创建提前 （与全局作用域一致）
  + 在函数中，不使用var声明的变量都会成为全局变量
  + 函数定义形参，相当于在函数中用var声明了变量
* Js中没有块作用域，如if 的大括号之内声明的变量，外面也可见（ES6之前）
* Js中的this
* 解析器在调用函数每次都会向函数内部传递一个隐含参数this
  + This指向一个对象，这个对象我们称之为函数执行的上下文对象
  + 根据函数的调用方式的不同，this会指向不同的对象
    - 以函数的形式调用时，this永远是window对象
    - 以方法（自定义对象内的函数）的形式调用时，this就是调用方法的那个对象
    - 当以构造函数形式调用时，this就是新创建的对象
    - 使用call（obj）和apply(obj)调用时，this就是指定的那个对象
    - 在事件的响应函数（on类型和addeventlistner类型）中，响应函数是给谁绑定的，this就是谁
* Js中的构造函数（类）
* 创建一个构造函数，专门用来创建Person对象的
  + 构造函数就是一个普通的函数，创建方式与普通函数没有区别
  + 不同的是构造函数习惯首字母大写
* 构造函数与普通函数的区别就是调用方式不同
  + 普通函数是直接调用
  + 构造函数用new调用
* 构造函数的执行流程
  + 立刻创建一个新的对象
  + 将新建的对象设置为this，在构造函数中可以使用this来引用新建的对象，然后创建属性如 ： function Person(name ){ this.name = name;}
  + 逐行执行函数中的代码
  + 将新建的对象作为返回值返回
* 使用构造函数构造的对象称为一类对象，所以构造函数也被称为类
  + 通过构造函数创建的对象就是该类的一个实例
  + 可以使用 实例instanceof类 来判断实例是否是该类的实例
* 所有对象都是Object的后代实例
* Js中的原型
* 我们所创建的每一个函数（普通&构造），解析器都会向函数中添加一个属性prototype
* Prototype就是一个原型对象
* 如果函数作为普通函数调用prototype的话，它就没有任何作用
* 当函数以构造函数的形式调用时，它所创建的对象（实例）中都有一个隐含的属性
  + 我们可以通过实例的\_\_proto\_\_来访问该原型对象
    - 实例对象名.\_\_proto\_\_.属性
  + 所有该构造函数的实例所调用的原型对象是相同，都是构造函数的原型对象属性
    - 给原型对象中添加属性 ：构造函数名.prototype.属性 = “”
  + 所以原型对象就相当于该构造函数的所有实例的公共的区域
    - 我们可以将实例中共有的内容（属性，方法），统一设置到原型对象中
  + 使用in检查对象中是否含有某个属性时，如果对象中没有，但是原型中有，也会返回true
  + 可以使用hasOwnProperty（“属性”）来检查对象自身中（不包含原型中）是否含有该属性
  + 原型对象也是对象，所以也有原型
    - 当我们使用一个对象的属性或方法时，会先在自身中查找（原型链）
      * 如果自身有，就调用，
      * 如果自身没有，则去原型中找
      * 如果原型中还是没有，就去原型的原型中找，直到找到object对象的原型，
      * 如果object对象的原型对象还是没有找到，就返回undefined
* Js中的垃圾回收
* 当一个对象没有被任何一个变量或者属性调用的时候，就会浪费堆内存的空间，此时js拥有自动的垃圾回收机制，会自动将这些垃圾对象从堆内存中销毁
* 为了让js更快速识别哪些是垃圾，我们尽量将已经不再需要使用的对象设为null

# Js中的数组

* 数组也是一个对象，属于内置对象
* 创建数组
  + 使用构造函数：Var arr = new Array()
    - 可以同时给数组赋值 : var arr = new Array(1,1,1,2)
    - 如果只传一个参数时，如 ： var arr = new Array(10) 就会创建一个长度为10的数组，所以构造函数创建数组一般不用
  + 使用字面量：Var arr = []
    - 使用字面量可以同时给数组赋值：var arr = [1,1,1,2]
* 读取数组的元素：
  + Array[index]
  + 如果没有值，返回undefined
* 数组中的元素也可以是任意数据类型
* 数组的方法
  + Concat 拼接两个数组放回新数组
  + Push 在数组最后加一个元素，返回新数组的长度
  + Pop 删除数组最后一个元素，返回被删除的元素
  + Unshift 在数组开头加一个元素，返回新数组的长度
  + Shift 删除数组开头的第一个元素，返回被删除的元素
  + Slice（start,end）截取从start到end的元素封装到新数组中返回
    - 该方法不会改变原数组
    - End可以不写，直接截取start到数组末尾的所有元素
    - End可以传负值，从最后一个开始数
  + Splice（start,num） 可以删除数组中的指定元素
    - 使用splice会改变原数组，会将指定的元素从原数组中删除，并将被删除的元素封装到新数组中
    - 第三个以后的参数，可以传递一些元素，这些元素将会自动插入到开始位置索引前边
  + Sort(function(a,b){}) 参数可以使用回调函数，来制定排序规则
  + 浏览器会分别使用数组中的元素作为实参去调用回调函数
  + 使用数组的那个元素不确定，但是a肯定在b的前边
  + 浏览器根据回调暗示的返回值来决定排序
    - 如果返回一个大于0的值，则元素会交换位置
    - 如果大于0，则不会
    - 如果相等，则不做
    - 如果需要升序排列，就返回a-b
    - 如果需要降序排列，就返回b-a
* 数组的遍历：foreach方法
  + Foreach方法需要一个函数作为参数
    - 函数由我们什么不由我们调用的，称为回调函数
    - 数组中有几个元素，该函数就执行几次
  + Arr.foreach(function(){})
    - 回调函数由浏览器调用，调用时传递三个参数（如果有）
      * 第一个参数：元素的值
      * 第二个参数：元素的索引
      * 第三个参数 ：整个数组对象
  + 此方法方便，但是兼容性差，IE8以下用不了
* Array.from(document.queryselectorAll（）) 可以将dom元素的伪数组封装成数组
* Js中的call()和apply()
* 这两个方法都是函数对象的方法，需要通过函数对象来调用
  + Fun.call()
* 当对函数调用call().apply()时，都会调用函数执行
* 在调用call（）和apply()时，可以将一个对象作为第一个参数
  + 此时该函数执行时的this将指向这个对象
* Call()方法可以将被调用的函数的实参在对象之后依次传递
  + Fun.call(obj,a,b)
* Apply()方法需要将被调用的函数的实参封装到一个数组中统一传递
  + Fun.apply(obj,[a,b])
* Js中的argument
* 在调用函数时，除了传递this隐含参数，还有argument隐含参数
* Argument是一个类数组对象，它也可以通过索引来操作数据，也有获取长度的属性
* 在调用函数时，我们所传递的实参都会在argument中保存
* 我们及时不定义形参，也可以通过argument来使用传递的实参
  + Argument[0],argment[1]等等
* Argument.callee 属性可以获取当前 argument所在的函数的对象
* Js中各类内置对象
* Date对象
* 创建Date对象
  + Var d = new Date()
    - 此时间为该行代码被执行的时间
* 创建一个指定的时间对象
  + Var d = new Date(“28/03/2020 23:58:00”)
* 创建一个时间戳
  + Var t = d.getTime()
  + 从1970/1/1到现在所经过的毫秒数
  + 计算机底层在保存时间时都是适用时间戳的
* Math工具类
* Math不是构造函数，而是工具类
* Math的方法
  + Math.abs()
  + Math.celi()
  + Math.round()
  + Math.floor()
  + Math.random()
    - 生成X-Y之间的随机数 ： Math.random()\*(y-x)+x
  + Math.max(a,b,c)
  + Math.min(a,b,c)
  + Math.pow(x,y) : x的y次方
  + Math.sqrt() 算根
* 包装类
* String() :可以将基本数据类型转为String对象
* Number() ：可以将基本数据类型转为Number对象
* Boolen() ： 可以将基本数据类型转为Boolean对象
* 注意： 在实际开发过程中不会使用该类包装类对象，会在比较的时候会产生不可预期的结果
* 包装类存在的意义 ：当我们对基本数据类型的值去调用属性和方法（如toString()）时，浏览器会临时使用包装类将其转换为对象，然后调用对象的属性和方法。
* String对象
* 在底层，字符串是以字符数组的形式保存的
  + 所有有Length,索引之类的数组操作方法
* CharAt(索引)
* charCodeAt() 获取指定位置的字符编码 unicode编码
* fromCharCode（通过String构造函数调用）可以根据字符编码去获取字符 unicode编码
* slice(开始位置，结束位置)
* substring(开始位置，结束位置) 跟slice几乎一模一样，区别：slice结束位置可以传递负值
* split(“-”） 根据-来截取字符串进一个数组
  + 如果传空字符的话，会将每个字符装进数组
* 正则表达式的对象
* 创建对象：
  + Var v = new RegExp(“正则表达式”，“匹配模式”)
    - 构造函数可以动态的拼接正则表达式。
  + 使用字面量 : var reg = /正则表达式/匹配模式
* 方法
  + Test()
* 正则表达式
  + | 或
    - a|b ： a或b
  + [] 或
    - [ab] ： a或b
    - [a-z] ： 任意的小写字母
    - [A-Z] ： 任意大写字母
    - [A-z] : 任意字母
    - a[bcd]e : abe ace ade
  + [^] 除了
    - [^A-Z] ：除了大写字母
  + {} 量词
    - a{3} ：找a连续出现 三次
    - a{1-3} ： 找 a连续出现 一次到三次
    - a{1,} : 找a出现一次以上
  + （）分组
    - (ab){4} : 找ab连续出现4次
  + + 量词
    - 相当于{1,}
  + \* 量词
    - 相当于{0,}
  + ? 量词
    - 相当于{0，1}
  + ^ 开头
    - 跟[^]完全不一样
    - /^a/ ： 找开头是a
  + $ 结尾
    - /a$/ ：找结尾是a
  + . 表示任意字符，除了换行
  + \ 转义字符
    - \. : 表示.
    - \\ ：表示\
  + \w 表示任意字母，数字和下划线
  + \W 除了字母，数组，下划线
  + \d 任意数字
  + \D 除了数字
  + \s 空格
  + \S 除了空格
  + \b 单词边界
    - /\dchild\d/ ：找是否有单词child 如果是children也是false
  + \B 除了单词边界
* 匹配模式
  + i 忽略大小写
  + g 全局匹配模式
* 配合String对象：
  + Str.split(/[A-z]/) : 根据所有字母拆分字符串
    - 此方法默认全局匹配
  + Str.search(/a[bc]1/) ：查询字符串中是否有ab1 或者 ac1 并且返回第一个找到的字符串的索引
    - 不能设置全局，只能查找第一个
  + Str.Match() : 根据正则表达式，从一个字符串中将符合条件的内容提取出来
    - 默认情况下没有全局
    - Str.Match(/[a-z]/ig)： 将匹配到所有字母封装进数组后，返回，及时只匹配到一个结果也会封进数组
  + Str.replace() : 被替换的内容可以接受一个正则表达式作为参数
    - 正常用法 ： str.replace(“需要被替换的内容”,””) ：用第二个参数替换str中的第一个参数（通常是正则表达式）
    - 默认情况下没有全局
    - 返回新字符串

DOM

1. JS通过DOM操作网页HTML
2. 节点：Node ,构成网页的最基本的单位
   1. 文档节点：HTML
   2. 元素节点:HTML的标签
   3. 属性节点 ： HTML标签的属性
   4. 文本节点 ： 文本

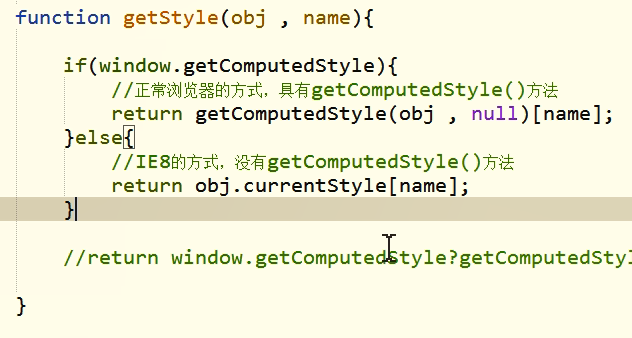
DOM中的属性和方法

1. 查询
   1. DOM中有属性BODY 可以直接获得BODY对象： document.body
   2. DOM中的属性html 可以直接获得html ： document.documentElement
   3. DOM中的属性 all 可以获得所有的元素的类数组
   4. DOM中的方法： document.querySelector()
      1. 可以根据CSS选择器来查询一个元素节点对象 ：
      2. 虽然ie8没有getElementByClassName ，但是可以用此方法
      3. 此方法只会返回一个元素，即使满足条件的元素只有一个，也只会返回第一个
   5. 方法： document.querySelectorAll() 解决上面（c）的问题，即使只返回一个元素，也会放在数组里面
2. 增删改方法：
   1. 创建元素节点： createElement(“标签名”)
      1. InnerHtml也可以实现
   2. 创建文本节点 ： createElement(“文本内容”) （几乎不用）
      1. InnerHtml也可以实现
   3. 添加节点： 父节点.appendChild(子节点)
   4. 在指定的子节点前插入新的子节点： 父节点.insertBefore(新子节点，旧子节点)
   5. 替换节点 ：父节点.replaceChild(新节点，旧节点)
   6. 移除节点 ：父节点.removeChild(子节点)
      1. 如：子节点.parentNode.removeChild(子节点)

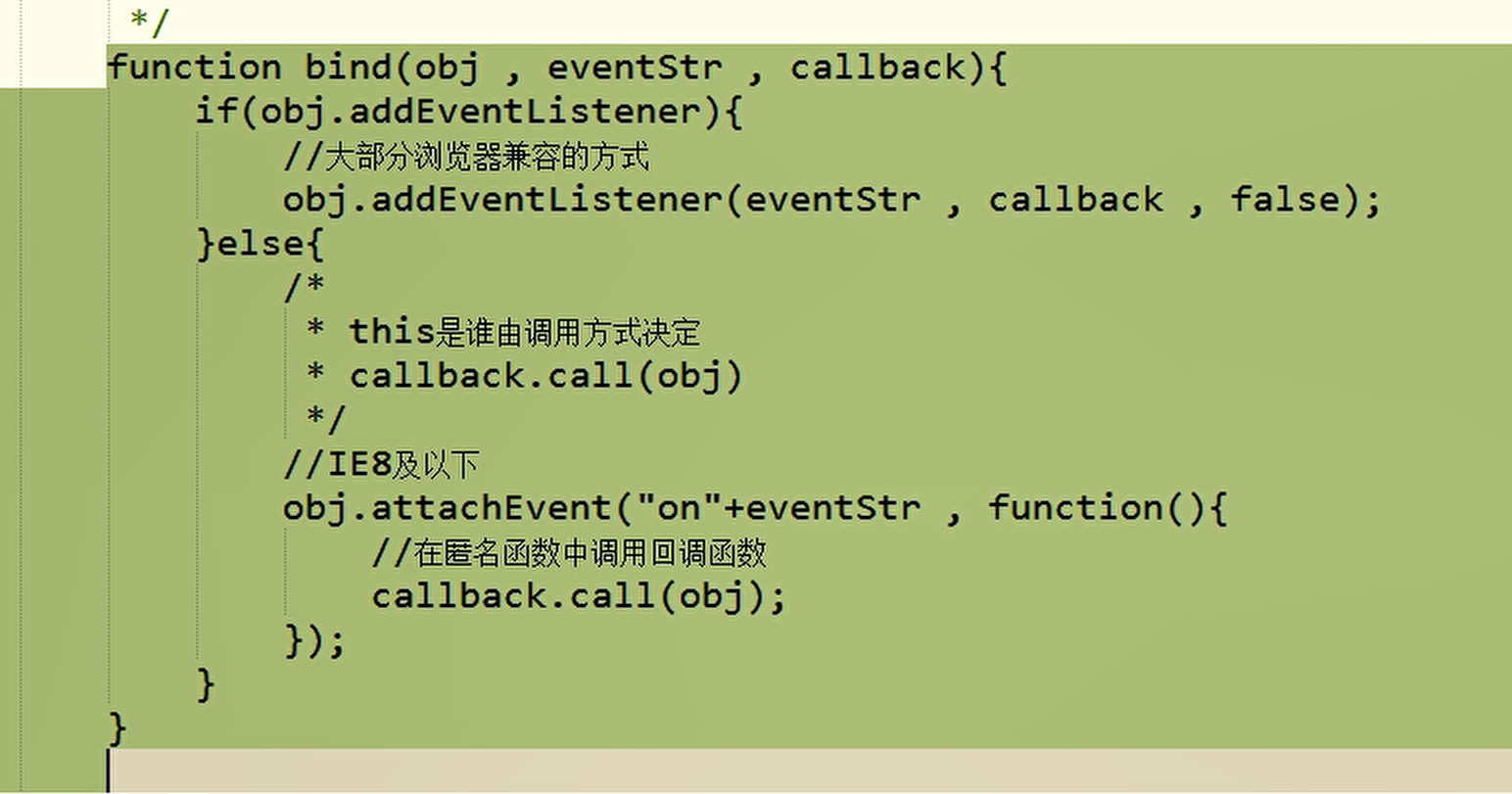
添加删除记录实战

1. 事件的回调函数称为响应函数
   1. 响应函数里return false可以取消a标签，button标签 的默认功能
   2. 在事件的响应函数中，响应函数是给谁绑定的，this就是谁
2. Confirm（”确认要删除吗”） 可以调出提示框，返回值为boolean
3. 响应函数在for循环里，是异步的，for循环在页面加载的时候早已经执行完毕，所以响应函数里无法用到for循环里的i,此时i已经是最大索引+1了
   1. 如果需要调用响应函数的对象，直接使用this，而不是用数组[索引]

使用DOM操作CSS

1. 通过js修改元素的样式(唯一可以修改元素的属性，下面其他属性都是只读)：
   1. 语法：元素.style.样式名= “样式值”
   2. 通过DOM的style都是修改或者读取内联样式，所以优先级都是比较高的
   3. 如果在html样式中写了!important 在js中修改了样式也起不了重用，所以慎用
2. 在CSS中的带-的样式名，在JS中变为驼峰
3. 通过js获取当前元素的现实的样式（只读）
   1. 元素.currentStyle.样式 ：读取的是当前正在显示的元素的样式（只有ie支持）
   2. GetcomputedStyle() 这个方法来获取当前的样式（兼容性高）
      1. Window的方法，可以直接调用
      2. 需要两个参数 ： 第一个为要获取样式的元素，第二个传递一个伪元素，一般都传null
      3. 返回一个对象，该对象封装了元素的样式
         1. 可以通过对象.样式来读取样式
      4. 该方法不兼容ie8以下浏览器
   3. CurrentStyle getComputedStyle有兼容性问题，所以可以新加一个方法来判断具体在用户浏览器里，具体用哪个。
      1. 
      2. 上面getComputedStyle必须加window，否则ie8下会报错，因为函数作用域，全局作用域都找不到此方法时会报错，如果加了window找不到此方法只会出现undefined。（因为对象.属性不存在时只会出现undefined不会出错）
   4. clientWidth,clientHeight 获取元素的可见宽度和高度（包括内容区+内边距）（只读）
      1. 返回的值都不带px ，可以直接计算
   5. offsetWidth,offsetHeight :获取元素的整个宽度和高度（内容区+内边距+边框）
   6. offsetParent : 可以用来获取当前元素的定位父元素，
      1. 会获取离得最近的开启定位的祖先元素，如果所有祖先元素都没有定位，则返回body
   7. offsetLeft offsetTop :
      1. offsetLeft 当前元素相对于其定位父元素的水平偏移量
      2. offsetTop 当前元素相对于其定位父元素的垂直偏移量
      3. scrollWidth,scrollHeight : 获取元素的滚动区域的高度和宽度（与clientWidth和clientHeight区别开）
      4. scrollLeft,scrollTop ： 获取滚动条滚动的距离
         1. 当满足element..scrollHeight- element..scrollTop==element.clientHeight 的时候，说明垂直滚动条滚动到底了
            1. 此方法可以用来做用户读取协议的打勾检验

Js中的事件

1. 各类常用事件
   1. Onscroll 滚轮移动时
   2. Onmousemove 鼠标移动时
   3. Onclick 点击时
2. 事件对象
   1. 当事件的响应函数被触发时，浏览器每次都会将一个事件对象作为实参传递进响应函数，该事件对象封装了事件相关的一切信息。比如朱建坐标，键盘哪个按键被按下了
   2. 在IE8及以下浏览器中，浏览器不会往响应函数里传event对象
      1. 所以经常在响应函数里这么写： event = event || window.event
      2. 上面代码表示，让实参event如果被浏览器传进来的话就等于自己，如果没有被传进来（IE8以下），就等于window.event
   3. ClientX ClientY 鼠标的坐标
   4. pageX pageY 表示相对于页面的鼠标距离（有滚动条的情况），而不是当前视口
      1. 不支持ie8
   5. Chrom的滚动条在body上，与其他浏览器(认为在html上)不一样，所以需要兼容判断
      1. st = document.body.scrollTop || document.documentElement.scrollTop
      2. 上面代码表示，如果body上有scrollTop就用，没有的话就用html的
   6. Target 触发事件的元素
   7. 事件的冒泡
      1. 事件的向上传导，当后代的事件被触发时，祖先的相同事件也会被触发
      2. 在大部分情况下，冒泡都是有用的
      3. 如果不希望冒泡，可以用后代元素的事件对象取消冒泡
         1. Event.cancelBubble = true;
   8. 事件的委派
      1. 将事件统一绑定给元素的共同的祖先元素。这样当后代元素上的事件触发时，会一直冒泡到祖先元素，从而通过祖先元素的响应函数来处理事件
      2. 委派是利用了冒泡
      3. 可以在祖先元素中利用事件对象的target属性来判断是哪个元素触发了事件，然后如果是期望的元素的话，再进行逻辑处理。
   9. 事件的绑定方式
      1. element.on\* 如 element.onclick
      2. element.addEventListener()
         1. 参数1：事件名字的字符串，不带on
         2. 参数2 ：回调函数
         3. 参数3 : 是不是在捕获阶段触发事件，需要一个布尔值，一般传false
         4. 使用此方法可以同时为一个元素的相同事件绑定多个响应函数。先绑定先执行
         5. 此方法ie8不支持
      3. Element.attachEvent()
         1. 参数1：事件的字符串，要On
         2. 参数2 ：回调函数
         3. 使用此方法可以同时为一个元素的相同事件绑定多个响应函数。后绑定先执行
         4. 支持ie8,不支持其他浏览器
      4. 手写band函数来兼容ie8和其他浏览器
         1. 
      5. 事件的传播
         1. 微软认为父子相同事件触发时，应该先触发当前元素，再触发祖先元素，也就是在冒泡阶段执行
         2. 网景认为父子相同事件触发时，应该先触发祖先元素，再触发当前元素也就是捕获阶段执行
         3. W3c综合了两个公司的方案
            1. 捕获阶段 ：从最外层元素传播事件到最内层，但不会触发事件
            2. 目标阶段 ：事件捕获到目标元素，捕获结束开始在目标元素上触发事件
            3. 冒泡阶段 ：事件从目标元素向他的祖先元素传递，依次触发祖先元素上的事件。
            4. 一般在addEvenListener传false来表明事件冒泡阶段
      6. Js的事件实战：（拖拽）
         1. 当鼠标按下时，开始拖拽 ：onmousedown
         2. 当鼠标按下且移动时，元素跟着鼠标移动 onmousemove
         3. 当鼠标松开时，元素留在当前位置 omouseup
         4. 拖拽完后必须要销毁onmousemove和onmouseup
         5. 拖拽时鼠标的箭头要保持不变
         6. 因为文字默认时可以触发onmousedown事件拖拽的，所以得 取消此事件的默认行为：return false；
            1. AddEventListener得用event.preventDefault()来取消事件的默认行为，但是ie8不支持这个属性。
            2. 所以为了ie8不报错,需要 event.preventDefault() && event.preventDefault()来兼容ie8

上面代码表示如果有event.preventDefault()来执行，没有就不执行，此套路可以用在用在 ”如果有…才…，没有也不报错”的情况

* + 1. 鼠标滚轮事件
       1. Onmousewheel 鼠标滚轮事件，火狐不支持该事件，在火狐中需要使用DOMMouseScroll，需要用addEventListener绑定
       2. Onmousewheel的event.wheelDelta可以表示滚轮的方向，
          1. 正数是向上滚，向下滚是负数
       3. 火狐中用event.detail 表示滚轮的方向
          1. 正数是向下滚，向上的正数。
       4. 鼠标滚轮实战：向上滚box变短，向下滚，box变长
    2. 键盘事件
       1. Onkeydown
          1. 如果一直按着某个键不松手，事件会一直触发
          2. 当连续触发时，第一次和第二次之间会稍微长一点，第二次开始都非常快的多次触发，防止误操作
       2. Onkeyup
       3. 键盘事件一般绑给能获取焦点的对象或者是document
       4. 可以通过event.keyCode来判断哪个按键被按下了。返回unicode的值
       5. 如果需要判断ctrl,alt,shift键是否被按下可以用event.ctrlKey，event.altKey event.shiftKey
       6. 在文本框输入内容，是onkeydown的默认行为，所以不能return false。
       7. 可以用keycode配合return false来禁止文本框禁止数字输入
       8. 实战：使用键盘控制div移动
          1. 进阶：配合定时器，让定时器控制速度，键盘事件控制方向来防止div第一次移动是卡顿（onkeydown事件的特性）

BOM对象

1. 浏览器对象模型：可以通过js来操作浏览器
2. BOM对象：
   1. Window
      1. 代表的事整个浏览器的窗口，同时window也是网页中的全局对象
      2. 方法:
         1. Alert
         2. Comfirm
         3. setInterval()
            1. 作用：定时调用，可以将一个函数每隔一段时间执行一次
            2. 参数：参数1是回调函数，参数2是时间（毫秒单位）
            3. 返回值：返回一个Number类型的数据作为定时器的名字，通过这个名字可以用clearInterval（）来清除定时器

注意不要将这个标识设成全局变量，不然多个定时器时会有隐患。

* + - * 1. 实战：简单的图片轮播效果，并且有开始，停止按钮

注意，开始按钮多次点击时，会多次开启定时器，需要先关闭上一个定时器，再开启新的定时器

* + - * 1. clearInterval()接受的参数如果不是定时器的返回值，那么什么也不做
        2. 实战：定时器自定义修改div的样式的move函数

参数：需要修改的div，需要修改的样式，修改到哪个目标值，修改样式时的速度，修改完样式后执行的函数（回调），

* + - * 1. 实战：轮播图

将所有图片放在同一个ul里面拼成长串

最后一张图片的后面再放一张第一张图片，来实现真正的轮播效果，从最后一张到最后面的第一张图片时，将图片的容器的left瞬间设为0来保证图片又从头开始轮播了。

* + - 1. setTimeout()
         1. 作用：延时调用，只会执行一次
         2. 参数：与定时调用一样
         3. 返回值：与定时调用一样
  1. Navigator
     1. 代表当前浏览器的信息，通过该对象可以来识别不同的浏览器
     2. 一般使用userAgent来判断浏览器的信息
        1. 此属性可以用来识别浏览器，从而判断在什么浏览器做什么事
  2. Location
     1. 代表当前浏览器的地址栏信息
     2. 如果直接alert此对象的话，能获得地址栏的信息
     3. 如果将该对象修改为一个其他路径，会加载这个路径的页面，并生成历史记录。
     4. Assign(“”)方法：与直接修改对象为一个路径的作用一样
     5. Reload()方法 ：用于重新加载当前页面，作用于刷新按钮一样，传递参数true,可以强制清除缓存
     6. Replace()，用于加载一个页面，无法生成历史记录
  3. History
     1. 代表浏览器的历史记录
     2. 由于隐私原因，不能访问具体的历史记录，只能操作浏览器向前或者向后翻页
  4. Screen
     1. 代表用户的屏幕的信息，可以获取用户的显示器的相关的信息

操作class属性

1. Element.className = “”
   1. 通过修改元素的class属性来修改样式比较便捷，能同时修改多个样式
   2. 可以一行修改多个样式，浏览器只渲染一次，性能比较好
   3. 用这种方式，可以让表现和行为进一步的分离
   4. Element.className += ‘ ‘ + newClassName //新增一个类名
2. 实战：二级菜单的展开关起效果。
   1. 需要一个开启后，其他的关闭的效果
   2. 需要有开起合上的动画效果
3. classList属性：
   1. 可以操作dom的class属性
   2. Element.classlist.add(“”) 给页面元素加class,可以一次加多个，如果class已存在也不报错
   3. Elment.classlist.remove(“”) 移除指定的class ，不存在也不报错
   4. Element.classlist.item(索引) 返回索引对应的class，不存在则null
   5. Element.classlist.contains(“”) 检查是否存在class，返回布尔值
   6. Element.classlist.toggle(“”) 在则删除，不存在则加，返回布尔值

Json

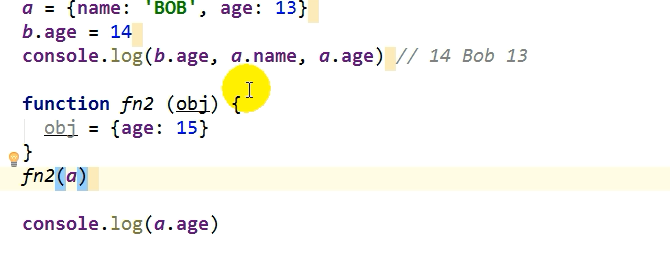
1. Javascript Object Notation ： JS对象表示法
2. Js中的对象只有js自己认识，所以需要将js对象转成大家都认识的字符串，这个时候就需要Json
3. Json就是特殊格式的字符串，可以被任意的语言给识别，并且可以转换成任意语言中的对象，JSON在开发中主要用来数据交互
4. Json分类：
   1. 对象 ‘{“name”:“kaki”，“sex” : “man”}’
   2. 数组 ‘[”name”，“sex”]’
5. Json中允许的值：
   1. 字符串
   2. 数值
   3. 布尔值
   4. Null
   5. 普通对象（不能是函数对象）
   6. 数组
6. 将json字符窜转换为js中的对象：JSON.parse(json字符串)
7. 将JS对象换成json字符串 ：JSON.stringify(对象)
8. 因为IE7以下不支持JSON对象，所以可以通过引用外部的JSON的工具类文件，来创建JSON对象。
9. Eval():这个函数可以用来执行一段字符串的js代码，并将结果返回
   1. 但是此方法性能差，有安全隐患，所以不推荐使用。

# JS高级

数据类型方法

1. 数据类型
   1. 基本数据类型5种
      1. String
      2. Number
      3. boolean
      4. Null
      5. undefined
   2. 引用类型：
      1. 对象Object
      2. 数组Array ：有数值下标的对象
      3. 函数Function :可以执行的对象
   3. Typeof ：返回数据类型的字符串表达 如：type of undefined数据类型会返回“undefined”
      1. 如： Console.log(type of a === “number”) 如果a是Number的话就true；
      2. Typeof 可以判断下列类型
         1. String
         2. Number
         3. Undefined
         4. Boolean
         5. Function
      3. 不能区分object与array ,也不能区分object与null
   4. Instanceof
   5. === / ==

数据，变量，内存

1. 下面情况a的age是多少？
   1. 
   2. A对象不变，所以还是13
   3. 在js调用函数时传递的变量参数，是值传递还是引用传递？
      1. 理解1 ：都是值传递（基本/地址值）传递
      2. 理解2 ：可能是值传递，也可能是引用传递（地址值）
   4. ’js引擎如何管理内存
      1. 内存生命周期
         1. 分配小内存空间，得到他的使用权
         2. 存储数据，可以反复进行数据操作
         3. 释放小内存空间
      2. 释放内存
         1. 局部变量 ：函数执行完自动释放
         2. 对象：称为垃圾对象 => 垃圾回收器回收
         3. 下面代码，函数执行完后，b会自动释放，b的对象会进入垃圾回收器
            1. 

对象

1. 对象定义：多个数据的封装体
2. 为什么要用对象
   1. 统一管理多个数据
3. 对象的组成
   1. 属性 : 属性名（本质上都是字符串）和属性值（任意类型）组成
   2. 方法 ：特殊的属性（属性值是函数）
4. 对象属性的调用方法
   1. Obj.属性名
   2. Obj[“属性名”]
      1. 下面情况必须用此方式
         1. 属性名包含特殊字符： - 空格
         2. 通过变量调用对象属性

函数

1. 定义：实现n条语句的封装体
2. 为什么要使用函数：
   1. 提高代码的复用率
   2. 便于阅读交流
3. 如何定调用函数
   1. Function() :直接调用
   2. Obj.function() 通过对象调用
   3. New test() 通过new调用
   4. Function.call(obj) / function.apply(obj) 临时让function成为obj的方法进行调用，this指向obj

回调函数

1. 回调函数满足下面三种情况
   1. 你定义的函数
   2. 你没有调用此函数
   3. 但是最终它执行了
2. 常见的回调函数
   1. Ajax请求回调函数
   2. 生命周期回调函数

IIFE

1. 全称 ： immediately-invoked function expression
2. 作用：隐藏实现，不会污染外部的命名空间，可以用来编写js模块
3. 例子：
   1. (function(){ console.log(“匿名函数隐藏实现”)})()

Js中的分号问题

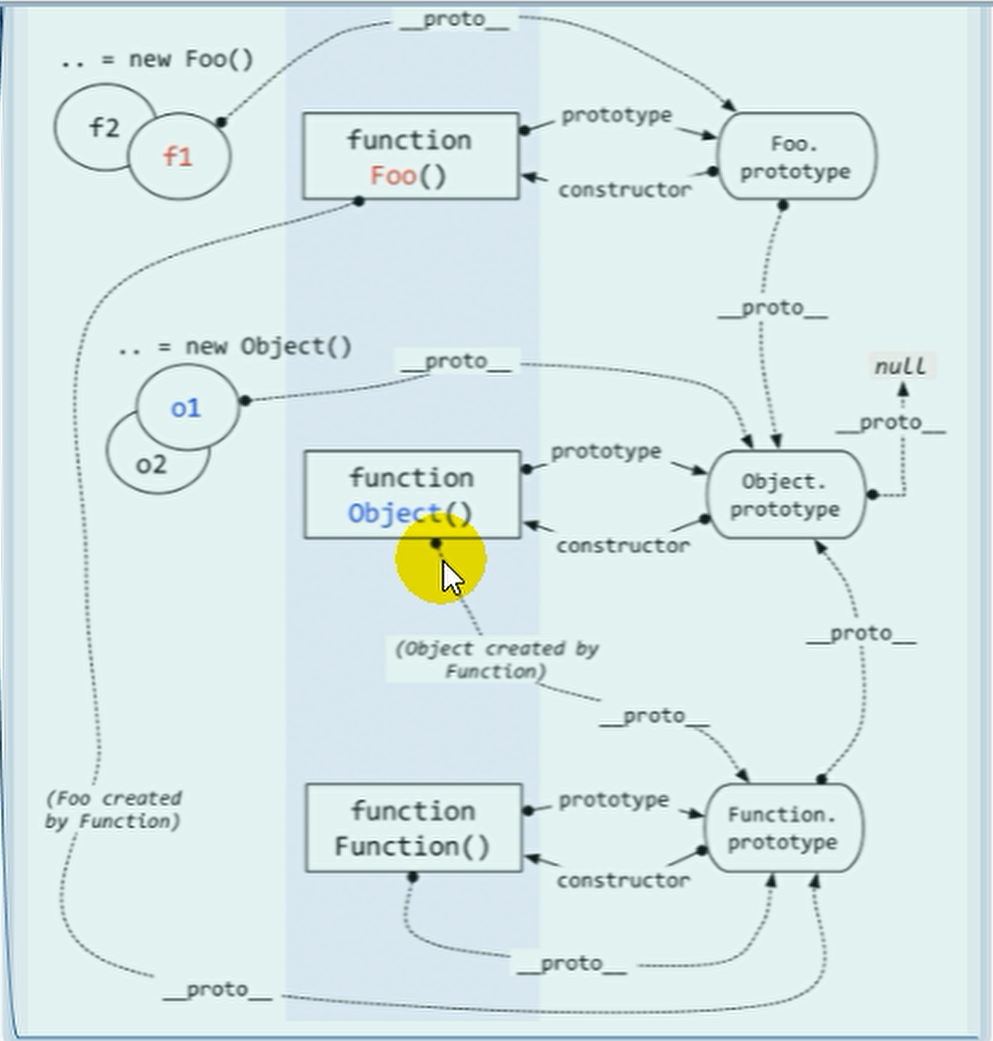
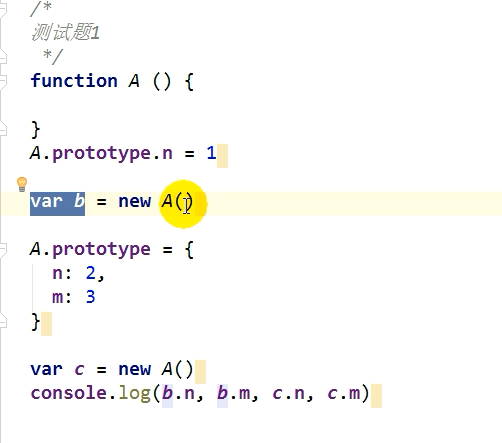
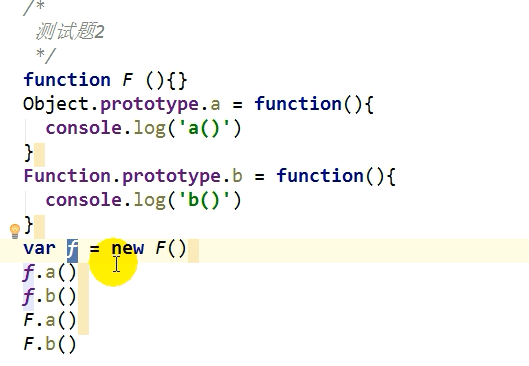
1. Js一条语句的后面可以不加分号
2. 在下面2种情况下不加分号会有问题
   1. 小括号开头的前一条语句 （IIFE的时候小括号开头）
   2. 中括号开头的前一条语句 （直接定义数组且不赋值的时候）
3. 解决办法：在小括号开头，中括号开头的时候加分号。

原型对象

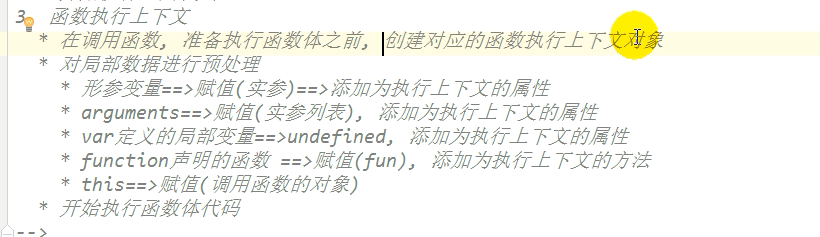
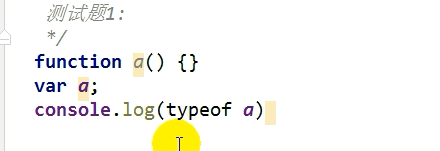
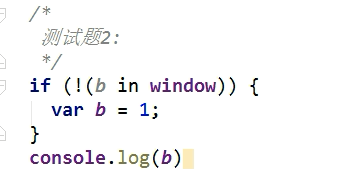
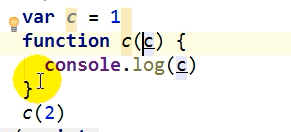
1. 每个构造函数都有一个prototype属性，它默认指向一个object的空对象实例（即原型对象）
   1. 构造函数的原型对象中有一个属性constructor，这个属性指向构造函数
      1. 也就是说每个构造函数都有指向原型对象的prototype属性，然后这个原型对象又有一个指回构造函数的constructor属性。
2. 每个构造函数的prototype属性为显示原型属性
   1. 定义构造函数时，自动添加的，默认是空obejct对象实例
3. 每个构造函数的实例对象的\_\_proto\_\_属性为隐式原型属性
   1. 定义实例时，自动添加，与构造函数的prototype属性指向同一个原型对象（内存栈中地址值相同）
4. 实例对象的隐式原型的值为其对应构造函数的显示原型的值

原型链。

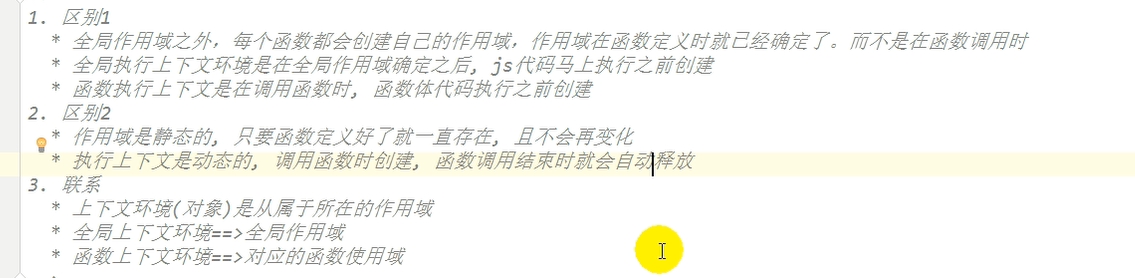
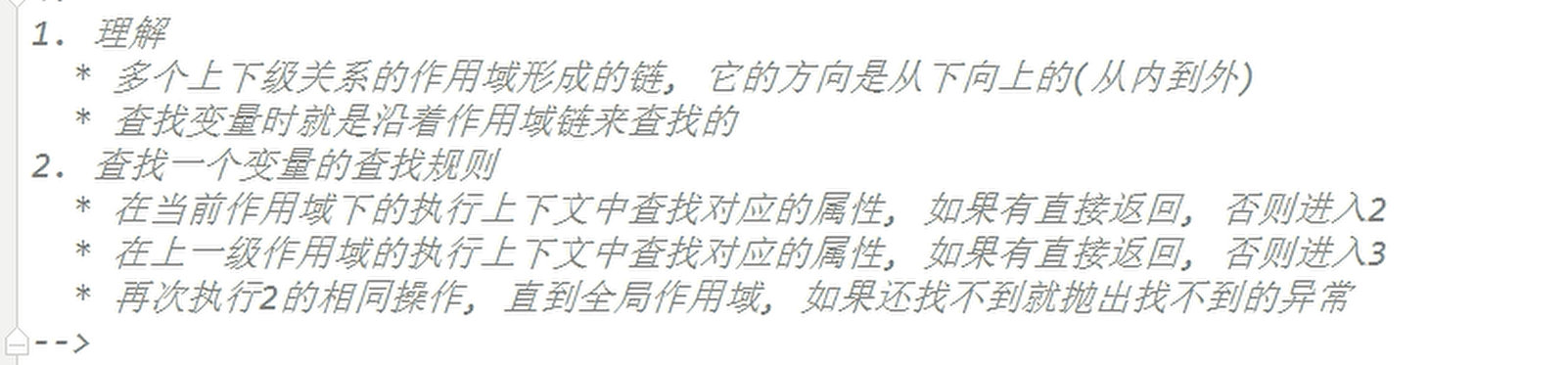
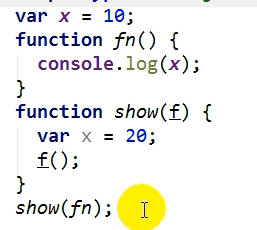
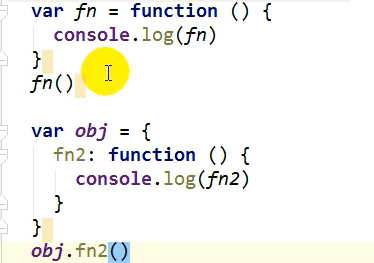
原型链

1. 访问一个对象的属性时，先在自身属性中找，找不到去原型对象找，再找不到去object的原型对象中找，再找不到就返回undefined了。
   1. Object的原型对象是原型链的尽头
2. 所有的构造函数的显示原型指向的对象默认是object的实例空对象（除了object）
3. 所有的构造函数的隐式原型都是Function的显式原型（包含Fcuntion构造函数，和object构造函数）
   1. 即所有构造函数都是Function的实例
4. Instanceof 的判断依据
   1. A instanceof B的情况下，如果B的显式原型在A的原型链上的话，那么就返回TRUE
5. 
   1. 上图是整个原型链，十分重要！！
6. 面试题
   1.  输出 ：1 undefined 2 3
   2.  输出：除了f.b()报错，其他都不报错。

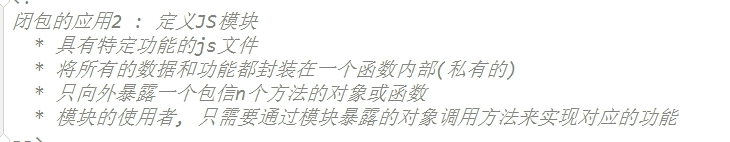
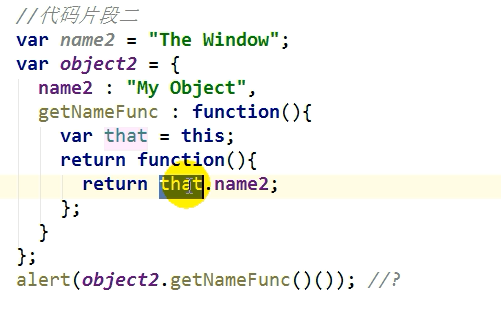
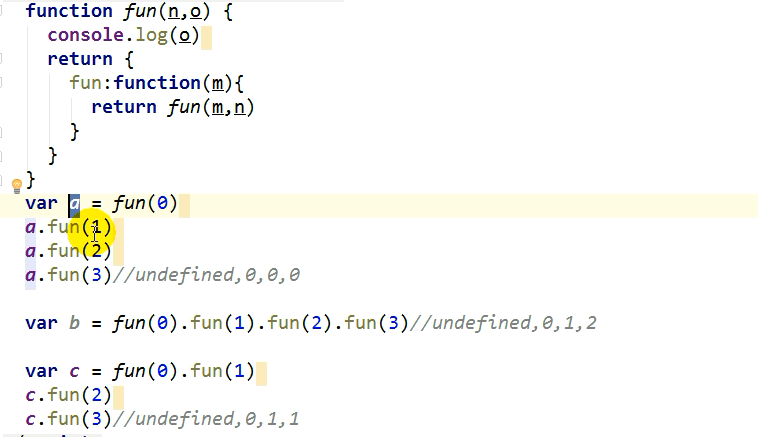
执行上下文

1. 全局执行上下文
   1. 在执行全局代码前将window确定为全局执行上下文
   2. 对全局数据进行预处理
      1. Var变量🡺提升，添加为window的属性
      2. Function声明的全局函数（非var声明的匿名函数）🡺赋值，添加为window的方法
      3. This 🡺 赋值（window）
   3. 开始执行全局代码
2. 函数执行上下文
   1. 
   2. 调用函数之前产生一个函数执行上下文对象
3. 在全局代码执行前，js引擎就会创建一个栈（后进先出）来存储管理所有的执行上下文对象（全局+函数的执行上下文）
   1. 多个执行上下文，栈的底部永远是全局执行上下文
4. 面试题：
   1.  输出 “function”
   2.  输出 ： undefined
   3.  输出： 报错。找不到function；

作用域与作用域链

1. 作用域与执行上下文的区别
   1. 
2. 作用域链
   1. 
3. 面试题：
   1.  输出：10
   2.  输出 ：报错

闭包

1. 当一个嵌套的内部函数引用了外部函数的变量或者函数时，就产生了闭包
2. 闭包到底是什么？
   1. 理解一：闭包就是嵌套的内部函数（大部分人）
   2. 理解二：包含被引用变量的对象（极少数人）
3. 产生闭包的条件：
   1. 函数嵌套
   2. 内部函数引用了外部的函数的变量或函数
   3. 执行内部函数的定义（不需要调用）
4. 判断产生闭包的次数
   1. 调用自己外部函数就产生几次闭包。即执行几次内部函数的定义就产生几次闭包
5. 闭包的作用：
   1. 让函数内部的变量在函数执行完后，仍然存活在内存中（延长了局部变量的生命周期）
   2. 让函数外部能直接访问函数内部的局部变量，并且操作
   3. 函数内部的变量在执行完函数后就会销毁，但是存在于闭包中的变量可以存活。
6. 闭包的生命周期：
   1. 产生：在嵌套的内部函数执行完时产生（非调用）
   2. 死亡：在嵌套的内部函数称为垃圾对象时（如：函数外部被赋值为null）
7. 实战：自定义Js模块
   1. 
8. 闭包的缺点与解决方式
   1. 函数执行完后，函数内的局部变量没有释放，占用内存时间会变长
   2. 容易造成内存泄露
   3. 解决方法：
      1. 能不用闭包就不用。
      2. 及时释放。
9. 面试题：
   1.  输出：the window
   2.  输出：my object
   3.  主要理解闭包有没有多次产生。

对象创建模式（5种）

1. Object构造函数模式
   1. New 空object实例，再动态添加属性/方法
   2. 使用场景：起始时不确定对象内部数据
   3. 问题：语句太多
2. 对象字面量模式
   1. 使用{}，同时指定属性/方法
   2. 场景：起始时，对象确定
   3. 问题 ： 如果创建多个对象，代码重复
3. 工厂模式
   1. 通过工厂函数动态创建对象并返回
      1. 工厂函数就是在函数内加一个对象属性，通过参数赋值给对象属性的属性，然后返回对象
         1. 
   2. 场景：创建多个对象实例
   3. 问题：对象没有一个具体的构造函数类型，都是object类型
4. 自定义构造函数模式
   1. 自定义构造函数，通过New创建对象
   2. 场景:需要创建多个类型确定的对象
   3. 问题：每个对象都有相同的数据，浪费内存
5. 自定义构造函数+原型组合模式
   1. 自定义构造函数，属性在函数中初始化，方法添加到原型上
   2. 场景：需要创建多个类型确定的对象

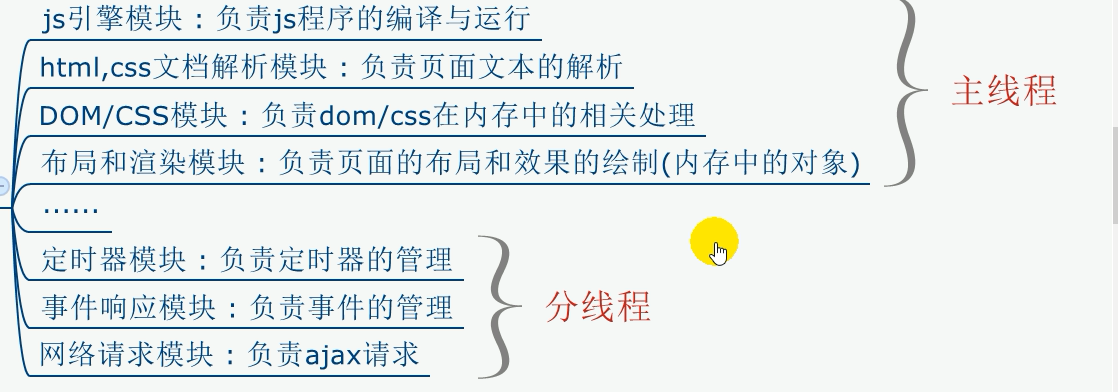
原型链继承

1. 利用原型链的机制，让自定义的构造函数实现类似java的类继承的机制
   1. 
   2. 子类型通过让构造函数的显示原型对象等于父类型的实例对象（默认的话，是等于Object的实例对象），来实现继承
      1. 此时需要再让子类型的原型对象（此时指向父类型的实例）的constructor(此时指向父类型的构造函数)指向子类型。
         1. Sub.prototype = new Super(); 子类型的原型指向将父类型的实例实现继承
         2. Sub.prototype.constractor = Sub; 子类型的原型（父类型实例）的构造器需要指回子类型

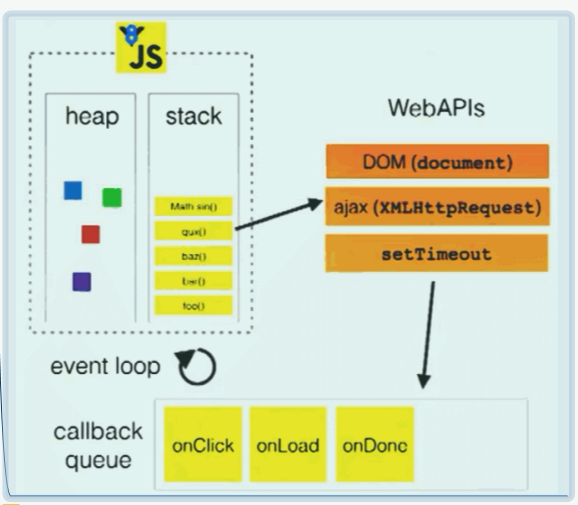
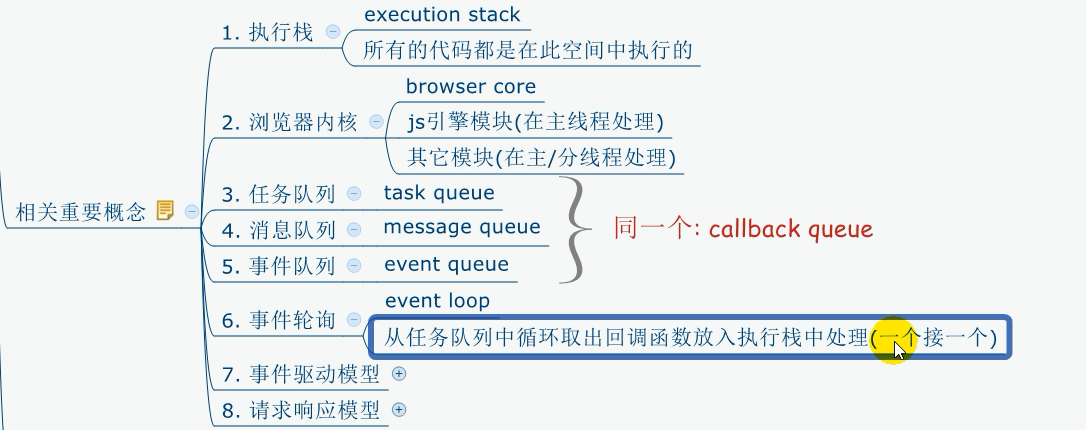
线程

1. JS是单线程运行的，但是H5中的Web Workers可以多线程运行
   1. 只能有一个线程去操作界面。
2. 浏览器是多线程
3. 浏览器是多进程（火狐是单进程的）

浏览器内核

1. 内核有很多模块组成
   1. 
2. JS引擎执行代码的基本流程
   1. 先执行初始化代码，包含一些特别的代码
      1. 设置定时器
      2. 绑定事件监听
      3. 发送ajax请求
   2. 然后在某个特定时刻执行回调函数（异步执行）

事件驱动模型

1. 
   1. 上图箭头表明了初始化代码执行后才会去执行回调函数
   2. 上图分为
      1. 初始化代码执行栈
      2. 事件管理模块（定时器管理模块/DOM/ajax）
      3. 回调队列
2. 

Web Workers

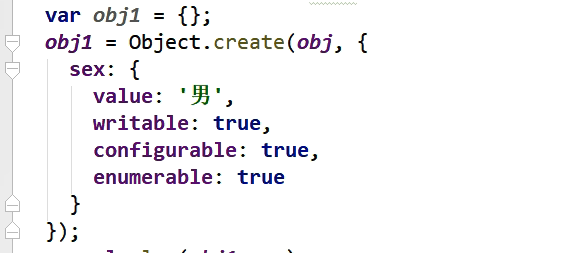
1. 
2. 使用workers(主线程上的js)
   1. 创建worker对象 ： var w = new Worker(“xx.js”) 参数传递分线程执行的js
   2. 向分线程发送消息 ：w.postMessage(数据)
   3. 绑定接受消息的监听事件 ： w.onMessage = function(event){ console.log(event.data)}
3. 使用workers(分线程上的js)
   1. Var onmessage = function(event){ var number = event.data; 处理number ；postMessage(number) }
4. Worker的缺点
   1. 不能跨域加载JS
   2. 不能操作DOM（因为分线程只有Worker对象，没有window对象）
   3. 不是每个浏览器都支持这个新特性
   4. 慢

# ES5

严格模式

* 1. 在全局或者函数的第一条语句定义为 ： “use strict”
  2. 作用：
     1. 变量必须用var声明
     2. 禁止自定义构造函数中的this指向window（防止出错）
     3. 给eval函数中的东西创建作用域（更加安全）
     4. 对象中严禁用重名的属性

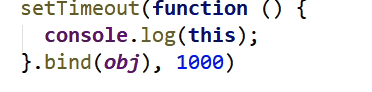
ES5中object对象方法扩展

1. Object.create(proyotype,{}) :
   1. 作用：指定当前对象的原型（第一个参数），然后创建当前对象的属性（第二个参数）
   2. 第二个参数可为一个对象，对象的每个属性是一个对象，该对象又有多个属性：
      1. Value:该属性的值
      2. Writable : 该属性是否可修改 默认为false
      3. Configurable : 该属性是否可以删除 默认为false
      4. Enumerable : 该属性是否能用for in枚举，默认为false
   3. 如：
2. Object.defineProperties(obj,descriptors)
   1. Object为构造函数
   2. 第一参数的object为需要扩展的对象
   3. 第二参数能添加属性，每个属性为对象形式，有get set的函数
      1. Get ：获取扩展属性的值 ，获取扩展属性值的时候，get方法自动调用
      2. Set ： 监听扩展属性，当扩展属性发生变化时候会自动调用，自动调用后将变化完的值作为参数传入方法
   4. 如：
   5. 此方法与对象本身的get 和set方法时一样的。
      1. 对象中本身就有get xxx(){} 和set XXX(){}方法

数组的拓展

1. Arr.indexof(索引)
2. Arr.lastindexof(索引) 倒着数
3. Arr.foreach(function(item,index){ })
4. Arr.map(function(item,index){ }) 遍历一个数组，进行操作，返回一个新的数组
5. Arr.filter(function(item,index){ }) 遍历过滤出一个新的子数组，返回条件为true的值
   1. 如过滤出大于3的数组 return item > 3

Call , apply, bind

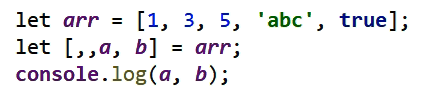
1. Call() 第二个参数开始，直接传入参数 立即调用
2. Apply() 将参数封装成数组，传入第二个参数 立即调用
3. Bind() 绑定完this（第一个参数传入后绑定），不会立即调用，而是将函数返回。 此时就可以自由自在的指定this了。广泛用于回调函数。
   1. 如 这边如果用call apply的话会立即调用，不用的话this会指向window ，只有用bind才能保证1秒后调用回调函数时，this指向obj

# ES6

Let，const

1. Let
   1. 特点：
      1. 在块作用域中有效
      2. 不能重复声明
      3. 不会预处理（找var 找function然后提前解析），不存在变量提升
   2. 应用： 遍历，循环监听 ：因为块级作用域中值都是唯一的，所以不需要转绑，也不需要闭包操作。
2. Const
   1. 特点：
      1. 无法被修改
      2. 其他特点同let

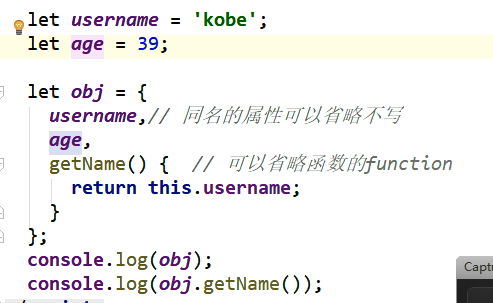
变量的解构赋值

1. let{xxx1,xxx2} = {xxx1 : “” ,xxx2 : ”” }
   1. 通过解构赋值向对象拿值批量赋值给let
   2. 可以用在参数上
   3. Function({username ,age}){ console.log(username,age)} 此时直接调用obj能直接解构obj给{username,age}，然后在function中打印
2. Let[a,b,c] = [1,2,3]
   1. 通过解构赋值向数组拿值批量赋值给let
   2. 数组根据下标位置取值。
   3. 如 a,b的值为5，abc

模板字符串

1. 把字符串放在 ` ` 中，然后用${}

简化对象写法

1. 同名的属性可以不写
2. 可以省略的function的字
3. 如

箭头函数

1. Let fun = () => xxxxxx
2. 形参的情况
   1. 没有形参的时候，小括号不能省略 let fun = () => xxx
   2. 只有一个形参的时候，可以把小括号省略 let fun = a => console.log(a)
   3. 两个及以上的时候，小括号不能省略， let fun = (a,b) => console.log(a,b)
3. 函数体的情况
   1. 函数体只有一条语句或者是表达式的时候，大括号可以省略，省略后会自动return 执行后的结果或者表达式的结果
   2. 函数体不止一条语句或者是表达式的情况，大括号不能省略
4. 箭头函数的特点：
   1. 简洁
   2. 箭头函数没有自己的this. 箭头的this不是调用的时候决定的，而是在定义的时候处在的对象就是它的this
5. 扩展理解：
   1. 箭头函数的this看外层有没有普通函数，
      1. 如果有，那么外层函数的this就是箭头函数的this。
      2. 如果没有，则this就是window

三点运算符（拓展运算符）

1. 用三点运算符收集的形参封装在真数组里面的
   1. Argument是伪数组
2. 三点运算符收集形参时必须放在最后，而且只收集剩下的形参。
   1. 如 function foo(a,…value){ }
      1. Foo(1,2,3,4 )此时的value为数组，数组中只有2，3，4
3. 三点运算符用在数组上
   1. …数组 可以直接分解数组成数组的每个值
      1. Arr = [1,2,3,4,]
      2. …Arr 得出1，2，3，4

形参默认值

1. Function foo (x= 1,y =2 ){ this. }
   1. 不传值就默认x = 1,y= 2

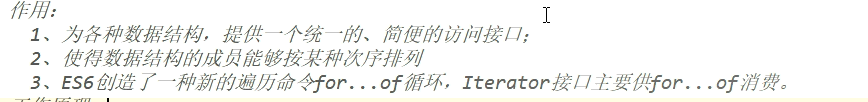
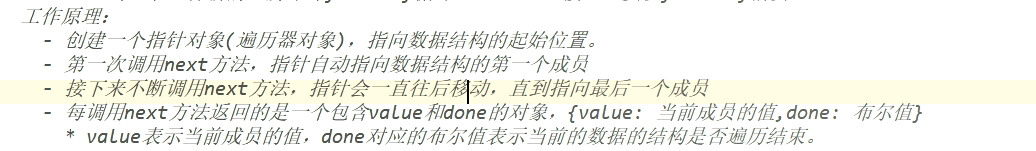
Promise对象

1. Promise对象：代表了未来某个将要发生的事件（通常是一个异步操作）
2. Promise可以将异步操作，以同步的流程表达出来，避免了回调地狱
3. Promise是构造函数，用来生成promise实例
4. Promise对象的3个状态
   1. 初始化 pending
   2. 成功 resolved
   3. 失败 rejected
5. 创建promise实例
   1. Let promise = new Promise((resolve,reject) =>{})
      1. Resolve函数表示成功后调用的函数，调用后promise修改为fullfilled成功状态
      2. Reject函数表示失败后调用的函数，调用后promise修改为rejected失败状态
   2. promise.then((data) =>{ 成功的回调 },(data)=>{ 失败的回调 })
      1. promise的then函数是在promise对象的异步任务执行完后立即执行的函数，
         1. 如果异步任务调用的是resolve()，那就执行成功的回调函数
         2. 如果异步任务调用的是reject()，就执行失败的回调函数
         3. 形参data是resolve(),reject()的实参 。
6. 实战：学完node后再看一遍视频

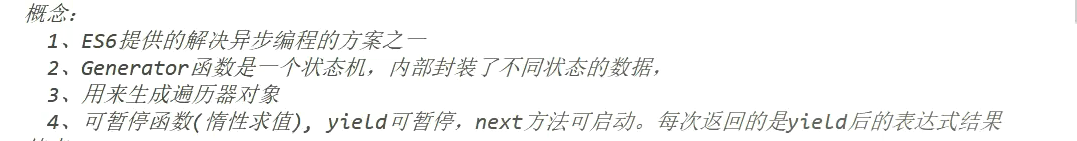
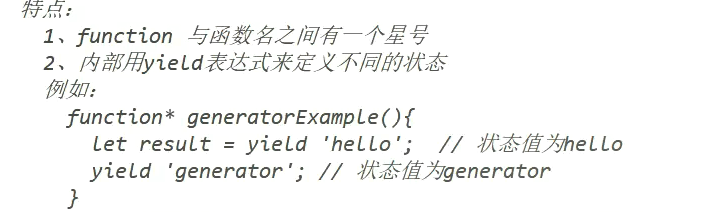
Symbol

1. Es6中的添加了一种原始数据类型symbol
2. 创建symbol属性值
   1. Let symbol = Symbol(); 调用Symbol()函数得到symbol值，（非构造函数，是函数）
      1. 参数可以传入字符串，来标识
   2. Symbol可以作为对象的属性名
3. Symbol()属性值是唯一的，解决命名重复的问题
4. For in , for of 无法遍历symbol属性
5. 内置Symbol值
   1. 除了自己定义使用的symbol值以外，es6还提供了11个内置的symbol值，指向语言内部使用的方法，可以进行读写（等同于在指定的数据内结构上部署本身存在或者不存在的方法）。
      1. 如：为本身没有迭代器接口的对象添加Symbol.iterator后，该对象就有一个iterator接口，但是此迭代器的实现得自己写。

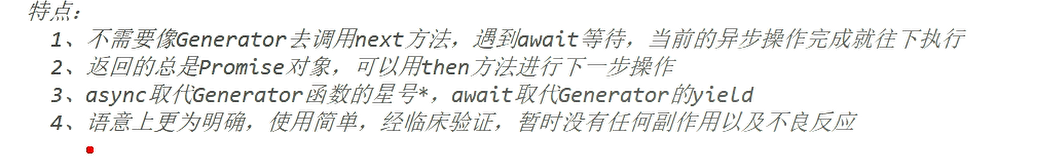
Iterator

1. 作用：
   1. 
   2. 目标身上有iterator接口才能使用for of
2. 工作原理：
   1. 
   2. 当遍历结束后，value返回undefined ，done返回true。
   3. 实战：用闭包自制迭代器（指针对象）
   4. 下列有iterator接口
      1. 数组
      2. 字符串
      3. Arguments
      4. Set容器
      5. Map容器
   5. 注意：对象没有Iterator接口！！！
      1. 可以为数据结构添加Symbol.iterator，然后用for of 遍历。如果没有迭代器就用for of的话，会报 is not iterabel的error
   6. 三点运算符和解构赋值，默认去调用iterator接口

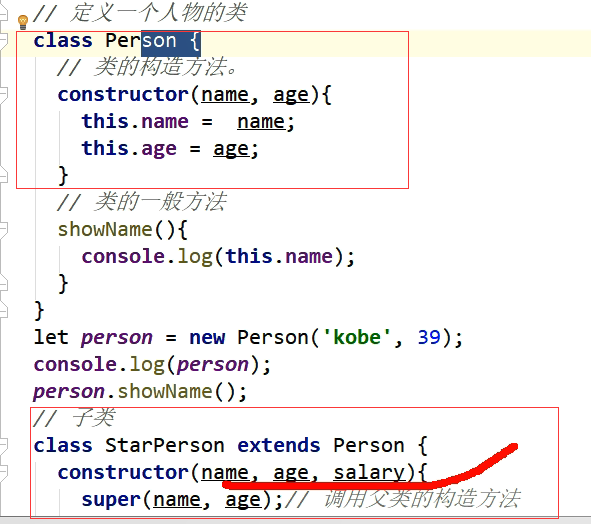
Generator函数

1. 
2. 特点
   1. 
      1. 如：function\* generatorExample() { yield ‘hello’ ; yield ‘generator ’ }
   2. Generator函数返回的是指针对象（迭代器），每次使用next()都会在yield暂停，然后返回一个对象{value : ‘xxx’ ,done : false or true}
      1. Value的值未yeild的表达式结果或者是返回值
      2. Done表示遍历是否结束了。
   3. Yield语句返回结果通常是undefined,但是，通过调用next()时传参，可以给yeild指定返回值
3. 实战：学完node.js后再回来做。

Async函数

1. 概念：真正意义上去解决异步回调的问题，同步流程表达异步操作
2. 语法
   1. Asunc function foo(){ await 异步操作； await 异步操作}
3. 特点：
   1. 
   2. Await的返回值：
      1. 如果是普通函数的话，就直接返回函数的返回值
      2. 如果是Promise.resolve() 需要在resolve中传入参数，作为await的返回值

Class类

1. 在类中通过constructor定义构造方法
   1. Constructor(name,age){ this.name : xxx ;this.age = age}
2. 通过class关键字定义类/l实现类的继承
   1. Class 子类 extends 父类
      1. 当子类声明构造方法的时候，必须使用super()来继承父类的构造方法
      2. 如：
         1. 
         2. 子类的显示原型的父类实例。
         3. 子类可以重写父类的方法（类似java）
3. 在类中定义普通方法：
   1. 不能用常规模式，只能用简写模式（有点像Java） 如：showName(){ return this.name}
   2. 类的一般方法都会放在原型上。实例的公共区域
4. 通过new来new出类的实例，与构造函数一样

字符串,数值的扩展

1. Str.Includes(str)
2. Str.startWith(str)
3. Str.endWith(str)
4. Str.repeat(count)
5. Num.isFinite(i) 是否是无限大的数
6. Num.isNaN(i)
7. Num.isInteger(i) 是否是整数
8. Num.parseInt(str) 可以转换到px的数字字符
9. Math.tranc(num) 干掉小数部分

数组的拓展

1. Array.from() 可以将伪数组转换为数组
2. Array.of(v1,v2,v3) 将一系列值转换成数组
3. Arr.Find(function(value,index,arr){return true}) 找出第一个满足条件返回true的元素
4. Arr.findIndex(function(value,index,arr){return true}) 找出第一个满足条件返回true的元素的下标

对象方法的扩展

1. Object.is(v1,v2) 判断2个数据是否完全相等(以字符串的形式判断)
   1. Object(0,-0) 等于false
   2. Object(NaN,NaN) 等于true
2. Object.assign(obj1,obj2,obj3..) 将源对象的属性赋值到目标对象上
3. ES6可以直接操作实例的隐式原型，来指定某个实例对象的原型是谁。
   1. Obj1.\_\_proto\_\_ = obj2 obj2是obj1实例的原型

深,浅拷贝

1. 拷贝数据：
   1. 深拷贝：基本数据类型：拷贝后会生成一份新的数据，修改拷贝以后的数据不会影响原数据
   2. 浅拷贝：拷贝的数据里有对象/数组： 拷贝后不会生成新的数据，而是拷贝引用，修改拷贝后的数据会影响原来的数据
2. 拷贝数据的方法：
   1. 直接赋值给一个变量 （浅拷贝）
   2. Object.assign() （浅拷贝）
   3. Array.prototype.concat() （浅拷贝）
   4. Array.prototype.slice() 浅拷贝
   5. Json.parse(Json.srtingify) 深拷贝 注意：此方法的拷贝的数据里，不能有函数，处理不了。
3. 浅拷贝（对象/数组）
   1. 特点：拷贝引用，修改新数据会影响原数据
4. 深拷贝：
   1. 特点：拷贝后生成完全新的数据，无论如何不会影响原数据
5. 如何实现深拷贝？
   1. 拷贝的数据里如果有对象/数组的时候就会造成只拷贝引用，所以是浅拷贝。
   2. 如果是想要深拷贝，可以继续拷贝需要拷贝的数据里的对象里的基本数据，一层一层拷贝下面，只拷贝基本数据类型，最终就会是深拷贝(递归)
6. 知识点储备：
   1. Typeof返回的数据类型只有6种 ：string,number,Boolean,function,object,undefined
   2. 用Object.prototype.toString.call( xxx).slice(8,-1) 可以返回更多的类型，如：Array,Null等
   3. For In循环，枚举对象时,i为属性名，数据时，i为下标
   4. 实战：自定义递归方法实现深度拷贝。

Set ,Map容器

1. Set : 无序不可重复的多个value的集合体
   1. 创建set
      1. Let set = new Set() or new Set([1,1,1,x,x,x]) 构造方法可以传数组，配合for of直接去重数组
   2. Set的方法：
      1. Set,add(xx)
      2. Set.delete(xx)
      3. Set.has(xx)
      4. Set.clear()
      5. Set.size
2. Map容器：无序的key不重复的多个键值对的集合体
   1. Let map = new Map([[ “aa” ,”11”]]) 用二维数组创建键值对
   2. Map.set(k,v) 添加键值对
   3. Map.delete(k) 删除
   4. Map.has(k)
   5. Map.clear()
   6. Map.size

指数运算符（幂） ： \*\*

1. 2 \*\*3 = 8

# JS模块化

1. 现在的js模块化都是用引入依赖的形式
   1. 、
2. 过去的js模块化用的历程
   1. 最早：普通的引入js文件，直接调方法
   2. 然后： 引用js文件，调用对象中的方法，避免命名重复，但是对象中是可以修改的，不安全
   3. 然后：IIFE模式：用匿名函数立即调用处理，将防止内部属性被修改
   4. 然后就是IIFE模式增强版：引入依赖（jquery）
3. 为什么要模块化？
   1. 降低耦合度
   2. 提高复用性
   3. 高可维护性
   4. 避免命名冲突
4. 模块化的问题：
   1. 项目太大，文件太多，顺序乱的话，难以维护
   2. 依赖模糊
   3. 请求过多
5. 模块化规范

CommonJS

* + 1. 说明
       1. 每个文件都可以当做一个模块
       2. 在服务器端：模块的加载时运行时同时加载
       3. 在浏览器端，模块需要提前编译打包处理
    2. 基本语法：
       1. 暴露模块：module.exports = value 或者 exports.xxx = value
       2. 引入模块：require(“xxx”)
          1. 第三方模块 ：xxx 为模块名字
          2. 自定义模块 ：xxx为模块文件路径

. 或者..开头

* + 1. 服务器端的应用 : node.js 网址： http://nodejs.cn/
    2. 浏览器端的应用 ： browerify 网址 <http://browerify.org/> 浏览器端的commomJS规范的打包工具
  1. AMD
     1. 说明
        1. 专门用于浏览器端，模块的加载时异步
     2. 语法：
        1. 定义暴露模块：
           1. Define(function(){return 模块}) 没有依赖的模块
           2. Define([“module1”,”module2”],function(m1,m2){return 模块}) 有依赖的模块
        2. 引入使用模块：
           1. Require([“module1”,”module2”],function(m1,m2){})
     3. 实现：require.js 网址：www.requirejs.cn

ES6规范

* + 1. 说明
       1. 依赖模块需要编译打包处理
    2. 语法：
       1. 导出模块 :export
          1. 常规暴露 （import的时候需要一个一个拿，挺麻烦的）

分别暴露：每一个数据结构前都加export

统一暴露：把所有定义完的数据结构统一放进对象中export

* + - * 1. 默认暴露 （大部分情况下就是给对象暴露）

语法：export default

Import的时候可以直接拿整个文件的模块，不需要再声明具体拿哪个模块

* + - 1. 引入模块：inport
         1. Import module1 from “路径”