## 第一周

### Js基本概念

JavaScript:一个面向对象的跨平台脚本语言

面向对象：分工协作

Js运行在各个浏览器，浏览器兼容各个平台

Js是一种弱类型语言

程序执行的过程中才会报错，是单线程，一次只能报一个错

程序一旦发生一个错误，程序将不会向下执行

在执行过程中会尽可能避免报错，语言的特性设计很松散

脚本本质就是编译型语言和解释性语言

编译型语言：将代码翻译成机器码

解释性语言：解析器及时解释

**小结：js特点**

1. 松散型即弱类型语言
2. 面向对象，但没有类的概念，基于原型的面向对象，一般用函数就行封装
3. 继承
4. 没有多态，但其本身却又天生多态

Js是鸭子类型语言

Js中所有的东西都是对象，表现为什么就是什么，不易原始形态定义，以表现形态定义

**Js架构：**ECMAScript

BOM:borwer object module

DOM：document object module

JS目标：操作浏览器，是浏览器端语言

操作的方面：显示，交互，请求

JS语言本身是区分数据类型的

数据是描述某个现象或精准储存某个操作

JS中只要表现形态不同，一定不是同一个东西

换行可以用br 或 /n

**Js基本数据类型：**

数字Number

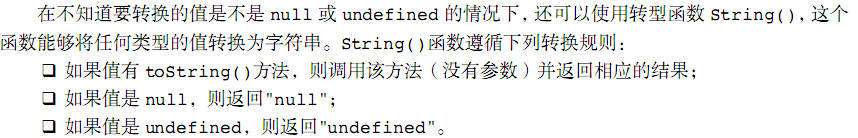
字符String

布尔值Boolean

特殊类型undefine null

undefined值是派生自null值，因此**undefined == null**

**null和undefined值没有toString（）方法**



Js不可以操作内存，只能根据浏览器解析机制被动的使用内存

关键字：单纯单词就有其功能

内存分成两种结构：栈 key：value形式 堆：二叉树形式

变量命名规范

1. 匈牙利命名法 ：

开头首字母是数据类型简写命名的

1. 驼峰命名法

每个单词开头字母大写

1. 下划线命名法

临时变量使用下划线

Var声明后在内存中开辟空间，var声明的变量都是undefined类型

如果变量声明后不赋值，就是undefined类型

**Typeof**

可以得到数据类型 typeof()typeof检测给定变量可的数据类型

**对一个值使用typeof操作符，返回的都是字符串**，“function”---则这个值是函数，”object”

---则这个值是对象或null

typeof操作符的操作数可以是变量，也可以是具体的数数值

注意，typeof是一个操作符而不是函数，因此typeof在使用时后面的圆括号写不写都可以

函数在ECMAScript中是对象，不是一种数据类型，因此可以通过typeof操作符来区分函数和其他对象

对未初始化和未声明的变量执行typeof操作都会返回undefined字符串，但是我们需要显示初始化变量，此时使用typeof返回“undefined”类型的变量一定是未声明的

**typeof在检测引用类型的值时用处不大，所有引用类型的值都是Object的实例**

**Instanceof**

Js中的instanceof用于判定原产地

例：{} instanceof Array 返回 false ； 【】instanceof Array 返回true

Function foo（）{} foo instanceof function 返回true

一般不用instanceof判断是否为函数类型，因为js中万事万物皆为对象，数组Array是Object的子集，使用instanceof判断是应该判断最小范围

**如果使用instanceof检测基本类型的值始终会返回false，因为基本类型不是对象**

**为什么typeof null结果是Object，而null instanceof Object 为false**

这是语言本身的bug，null是表示空的独立类型，null属于object类型，但是null不能被Object构造函数构建

**隐式类型转换**

解析器自发的进行了数据类型转换，这种转化叫做隐式数据类型转换，发生在数学运算中隐式数据类型转换是由浏览器中的js解析器转换的

加法中优先字符串类型，只要有一项是字符串，那么这个运算就从加法运算变成拼接运算任何数据类型和字符串拼接，都变成字符串类型

除了加法优先字符串外，其他的运算都优化数字，如果不是数字类型，试着转化为数据类型

**其他类型转换为布尔类型**

数字转换非零即为true

字符串转换非空即为true （空格也是字符，只有里面什么也没有才会为false）

特殊类型的变量undefined，null转换为布尔类型的变量后值都为false

非运算符！ 如果操作的变量不是布尔类型的都会进行强制类型转换为布尔类型变量

Number类型中任何非零数字值包括无穷大，都是true， 0和NaN转化为false

Object类型中，任何对象都是true， null是false

Undefined类型中的undefined值为false

**Parse解析**

parseInt（）任何数据类型转换为整数类型，从左到右一次检索遇到的每一个字符，如果有任意一个不为数字的部分，会删掉这个字符及后面的内容，如果第一个字符就是非数字，返回NaN。第一个空格不算，可以是+ 或- +代表正数，-代表负数

**parseFloat**：

从左到右依次检索遇到的每一个字符，如果有字符破坏了数字规则，就会删除这个字符及后面的内容parseFlot()会始终忽略前面的0，只能解析十进制数值，没有用第二个基数的方法

**Number（）**

进行整体的数据类型变换,只要存在任何不符合数字规则的部分，就会被转化为NaN

数值转换函数Number（）会将最前面的0忽略

Number（）函数 可以用于任何数据类型， 如果传入的是数字，则只是简单的传入和返回，遇到十六进制格式时会转换为相同大小的十进制数

**NaN**

强制把其他类型转换为数字，该类型根据方法规则实在无法创建，方法会返回NaN表示转换错误，但是尽力转换了。 判断NaN是否相等用**isNaN（）**

可以通过Number.NaN得到NaN的值，任何与NaN进行运算的结果均会为NaN,NaN与自身不相等（NaN不与任何值相等） **NaN的数据类型为number**

**toFixed()**

可把 Number 四舍五入为指定小数位数的数字 如：Num.toFixed(2)

利用toString（）可以将十进制转化为任意进制：

利用ParseInt()可以将任意进制转化为十进制

浏览器能识别的最大数字为八个字节999999999999999

特殊写法1e+n n表示1后面的零的个数

在js中if 条件{}中的变量无作用域，目前只有函数中有局部作用域

**什么时候用if else 和switch**

条件必须是具体值而不是范围，推荐使用swich

条件是范围推荐用ifelse

If中的括号中会进行隐式数据类型转换，但是在swich中的括号中的内容和 case中的比对是绝对比对===，如果数据类型不等，不会继续比对，如果可能存在不合法的参数，对参数预处理，即进行数据类型转换

Swich中会被穿透，break的作用就是阻止穿透

**三元运算符是if else的简写** boolean？trueValue：falseValue

**三大循环**

While()

For()

Do{}while()

Do while 比while多执行一次，多一个循环终止功能

For(var i = 0; i++ < 10; ){} 大括号内的i从1开始

**递归**

递归必须要有终止条件

递归的逻辑简单但性能比较差，当递归的深度不可控时不要使用

**break**

可以终止循环，break加上变量可以直接跳出多层循环 如：break x\_1，有个前提是这个变量需要加在指定for循环前边 x\_1:for()

**continute**

可以终止这一次的循环

### 错误类型

**SyntaxtError 表示语法错误**

**typeError:类型错误，把数据当成错误的类型使用**

**RangeError: 范围错误**

### 事件

**事件**：用户的操作导致我们在程序上可以做一些处理

在浏览器段JS是事件驱动的

事件是发生在标签上的

**点击事件**：

btn.onclick = function(){ //按钮点击事件

}

btn.ondblclick = function(){} //按钮双击事件

**鼠标事件**：

omousedown 鼠标按下事件 onmouseup 鼠标抬起 onmouseover 鼠标移入

onmouseout 鼠标移出 onmousemove 鼠标移动

**键盘事件**：onKeydown键盘按下 onkeyup键盘抬起 onkeypress键盘抬起

**元素事件**：onfocus 获取焦点 onblur 失去焦点 oninput 正在输入 onchange 已经改变

**窗口事件**：window 窗口的统一入口

onload 加载结束 onresize 窗口尺寸改变 onscroll 滚动条发生滚动



onmouseover会重复触发子集事件

onmouseenter不会

### 函数

函数：编程角度看，就是具有某种功能的代码判断

生活角度：函数就是一个待完成指定功能的工具

**函数的定义方式**

声明方式： function test（）{

//声明会被提前，可以在任何地方调用

}

赋值式： var test = function（）{

//只能在该函数以下位置调用

}

两种方式创建的区别：

JS是一个脚本语言，它的代码需要编译，编译过程中，所有的声明语句都会被提前

**函数的好处**

代码复用、封装、控制代码的执行时机

匿名函数，

function（）{

由于没有名字，该函数无法调用

}

(function(){

匿名函数定义的同时，立即执行，也叫做函数的自运行

})()

函数中的参数不做类型检查

Js中的全局变量的作用域是全局的，在代码的任何地方都可以定义，函数的参数和局部变量只在函数体内有定义，另外局部变量的优先级要高于同名的全局变量的优先级，可以在不同的函数中使用名称相同的局部变量

**基本类型是一个栈对应一个数据，一个变量名对应一个存储内容**

**引用类型有Object， function，Array，其中array是object的子集**

引用类型共有一个栈，这个栈之中可以放好多对象，可以把这个栈理解为一栋楼，每个房间都为一个独立的对象

变量声明提升：变量的声明不管写在哪里，都会被提升到该作用域的最前边，函数**提升的优先级别高于变量。变量声明会覆盖同名函数。变量名和函数名一样时，不管书写顺序，变量值会直接覆盖函数值，多个函数声明可以进行覆盖**

**Arguments**

Arguments对象包含了函数执行时传入的所有参数，arguments本身不是数组，而是经过封装的一个对象，它包含一个数组，即我们的参数列表。由于它可以当成数组来使用for循环遍历，我们管这类对象叫做伪数组

**Arguments用法**

它是一个伪数组，用来返回函数中的不定参数（用于不确定有几个参数的函数中），arguments用来储存函数调用时传入的所有参数。在函数不确定参数的情况下，在声明函数时不用谢函数括号中的形参

function foo(){

for(var i in arguments){

document.write(arguments[i]);

}

}

foo(1,2,3,4,5,6,7,8,9,"hello");

可以使用for循环遍历arguments数组

for（int i = 0; i < arguments . length; i++){

document.write(arguments[i]);

}

**遍历arguments高阶方法**

For（var i , value;value = arguments[i];i++）{

Document.write(value);

}

**在for循环发判断条件中，如果i等于数组的长度，伪数组arguments的第i个值不存在，则在arguments【i】中返回的是undefined类型，在for循环的条件判断中直接强制类型转化为布尔类型false，条件为假直接跳出循环**

要判断arguments的类型，使用typeof判断不准确，应该使用instanceof关键字进行判断

函数和变量基本一致，变量用来存储数据，函数用来存储代码

函数名代表功能，（）代表条件，{}内部存放代码

函数声明后在内存中开辟存储空间，将后面的自定义语句存放，等待执行

函数调用：函数名+（）表示执行函数大括号里面的所有代码

Tip：函数功能要单一；不要太庞大

函数的设计原则，**高内聚，低耦合**。

API就是语言本身提供给你实现某些功能的函数

如果直接写匿名函数会报错，因为匿名函数如果直接写无法调用，属于垃圾内容

**立即调用函数的写法**（function（）{}）（）

（）相当于把函数进行运算，也可以用+ -等运算符，其中！号称最快

立即调用函数失去了我们在适当时机调用的特性，立即调用函数的意义在于作用域

特殊写法：var foo = function poo(){ }

其中poo可以看做是一个有名字的匿名函数

1.可访问性，当具名函数给变量赋值的时候，具名函数的函数名只在函数内部可以访问，相当于一个局部变量

2.读写特性，具名函数的函数名是只读的

3.常量

**全局变量**

只要程序运行，全局变量会一直常驻于内存之中，导致程序占用内存较大，所以应尽量少用全局变量数据的存储是永久的，全局变量的永久是相对于程序的永久，

**局部变量**

在函数中声明的变量叫做局部变量；局部变量属于临时变量，用完之后就会被抛弃

参数机制

形式参数 相当于一个未赋值的变量

实参 实际参数（数据），给形参赋值

函数在运行之前就是字符串，在函数运行时在内存中开辟了一块空间，这块空间用来存储所有foo函数运行时需要用到的数据，把局部变量放入到当前空间，引用的时候在空间之中取出相应的内容进行使用，函数运行结束后内存有一个垃圾回收机制，1.计数法，2，标记法

**声明周期**：

全局变量声明周期是永久的，在程序的任何位置都可以访问

局部变量的声明周期是有限的，在作用域创建的时候被创建，在作用域被销毁的时候销毁

**JS中的糟粕特性**

声明提升：变量声明的提升是部分提升，函数声明的提升是整体提升

当你在任何位置编写一个变量声明的时候，无论在任何位置，当代码解析的时候，我们的解析会将这个变量的声明提升到当前域的最前列进行声明，只在原来位置保留赋值

在函数体内操作全局变量的时候，对全局变量赋的值也会影响函数体外的全局变量

**函数执行时会产生一个作用域**，当变量声明的时候会被解析器进行声明提升

函数没被调用之前永远都是字符串，函数只有被调用的时候被解析器解析为执行环境

**函数被调用时函数大括号中的代码就是执行环境**，只有函数在运行时才能创建执行环境

**用来储存函数内部的变量的值叫做变量对象**，以键值对的形式存数据的结构，

Function funcA(){

Var a = 100;

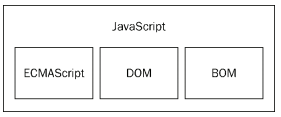
}

对于函数A来讲，它执行时会创建执行环境，需要存储执行时创建的数据，因此函数A也有自己的变量对象，这个变量对象在编译期间不存在，也就是说只有funcA执行时，才会创建一个变量对象，以键值对的形式存储变量a=100的值，当函数执行结束，执行环境被释放，则变量销毁，因此，变量对象只在函数活动期间存在，所以也叫活动对象。

**变量使用的就近原则**, 变量如果在函数内声明，根据调用原则，先查找自己的AO，即先调用函数内的变量，如果函数内部没有临时变量，继续向外查询，逐个作用域查询的机制，一直到顶层的全局作用域

基本类型是一个栈对应一个数据，一个变量名对应一个储存内容。

Javascript和ECMAscript通常用来表达相同的含义，但是js规定的功能比ECMAscript多的多，ECMAScript是js中的核心



**Js中区分大小写**

**Js中的变量可以保存任何类型的数据**，每个变量仅仅是一个用于保存值的占位符

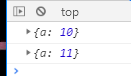
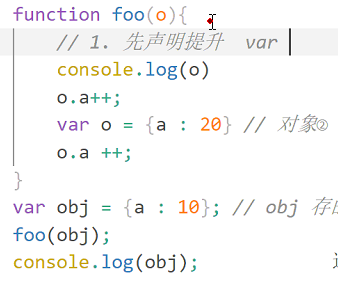
Var message = “hi”，初始化变量message但是并不会把它标记为字符串类型，初始化的过程只是给变量赋一个值，因此可以在修改变量的同时修改变量所保存的值的类型。

使用var操作符定义的变量是定义该变量作用域中的局部变量，不使用var定义的变量都是全局变量，在函数中定义个也是全局变量，但是给未经声明的变量赋值严格意义上会导致抛出referenceError错误。

实参永远是数值，传递的是变量，对象等都只能是数据类型，给形参（函数内部的局部变量）赋值的参数

形参相当于在函数体内声明的局部变量 形参===局部变量

**在函数中声明提升优先于实参给形参复制，如果同时存在声明提升和实参赋值，那么实参赋值有效**



**函数的参数传递**

Js中访问变量可以分为**按值、按引用**两种方式，而**参数只能按值传递**

function addTen（num）{

num+=10；

return num；

}

var count = 20；

var result = addTen（count）；

alert（count）； //20 没有变化

alert(result)； //30

这里addTen（）中的参数num实际上是函数的局部变量，传递的是count的值

立即执行函数**主要是创建作用域，作用：避免大规模大量使用全局变量**

**全局执行环境**被认为是window对象，因此所有全局变量和函数都是作为window对象的**属性和方法**创建，某个执行环境中的所有代码执行完毕后，该环境被销毁，保存在其中的所有变量和函数的定义也随之被销毁。

作用域：在js中函数大括号内部即为一个独立的作用域，在js中暂时没有块级作用域，现在学习的是ES3-ES5版本，但ES6中有块级作用域

局部作用域：因为程序在执行的时候会对函数进行特殊处理， 解析器会给函数创建独立的执行环境，根据执行环境创建一个独立的作用域（AO），这个活动对象用来存储临时的变量，局部作用域在执行结束之后会被内存自动清理

局部变量：函数开始执行时被声明，函数执行结束后被删除

作用域链

当代码在一个环境中执行时，会创建变量对象的一个作用域链，作用域链的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，作用域链的前端，始终都是当前执行的代码所在的环境变量对象，如果这个环境是函数，则将其活动对象作为变量对象，活动对象在一开始只包含一个对象，即arguments对象，作用域链的下一个变量对象来自包含（外部环境），再下一个对象来自下一个包含环境，一直延续到全局执行环境，全局执行环境的变量对象始终都是作用域链中的最后一个对象

**闭包**

定义：外部声明的函数在内部使用，吧变量返回出去，吧局部变量的声明周期延长了，闭包的前提是函数嵌套，外部函数声明的变量在内部函数被引用了，内部函数被返回到了全局，进行随时引用。

**闭包会引起内存泄漏，所以避免内存泄漏程序放在大型循环中**

每次外部函数执行，都会创建一个全新的活动对象AO，这个全新的活动对象内部的所有内容都是重置之后的。

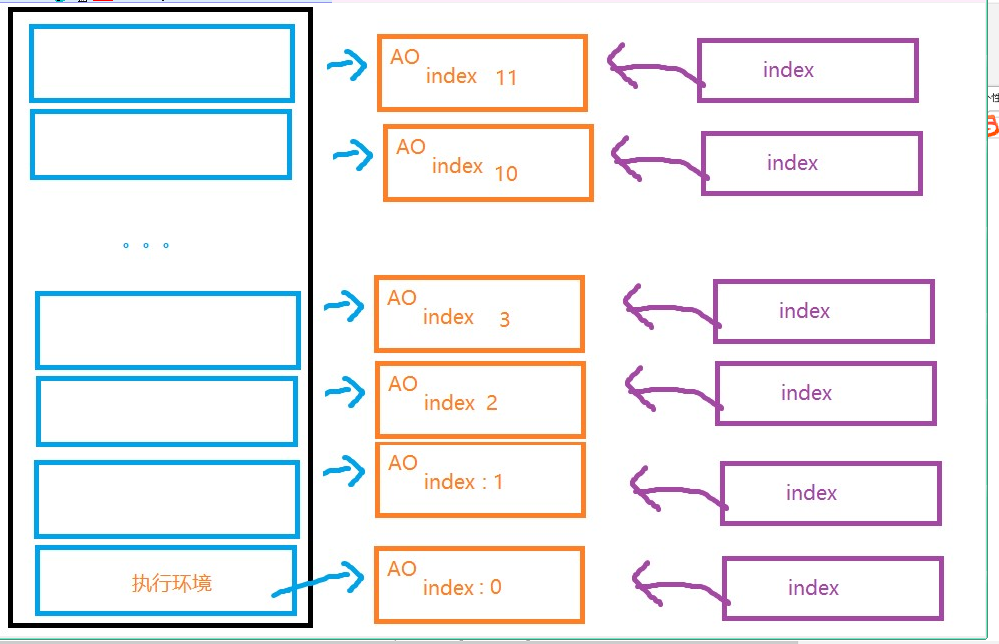
闭包可以保证变量的隐私性，局部变量的声明周期延长了

**闭包的前提：**

函数嵌套

1. 外部函数声明的变量，在内部函数被引用了
2. 内部函数被返回到了全局，进行随时引用

在闭包的产生过程中，外部函数执行完后之后，外部函数中的变量并不会被删除看，因为在内部函数中有一次这个变量的引用，但是内部函数被返回出去了，并没有调用到引用的这段代码，根据内存垃圾处理的标记机制，该变量有标记，在函数执行完之后标记并没有被删除，所以该变量没有被销毁



### 对象

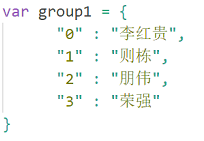
对象的本质是存储数据，同类别的数据放在一起集中管理

对象创建方式：

Var obj ={}

Var obj = new Object（）

数据在对象中以键值对的形式存在



**注意**：对象中的key键可以不加双引号，因为会发生隐式转换，但value值必须加双引号

对象中的取值方式 . 和 [ ]

Js运算中 . 运算 运算符仅次于（）， **. 后面的内容既不能是变量也不能是字符串**，而是直接写的单词或几个字母，**这个单词或字母表示对象中的key键**，是取值的简化方式

Js对象中的【】运算符表示取值，**【】里面是字符串**，表示对象中的key键

所有的key键都是字符串，声明时如果key值写的不是字符串也会默认转化为字符串

对象中的删除操作使用delete

Var obj ={a:10}

Delete obj a；

**基本类型与对象如何比对？**

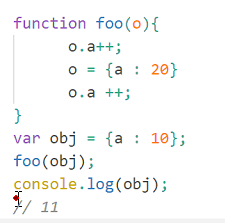
基本类型在变量中存放的是值，比对的时候是值比对，引用类型存的是地址，比对的时候是地址比对

五种基本类型的变量是**按值访问**的，因为可以操作保存在变量中的实际的值

引用类型的值是保存在内存中的对象，js不能直接操作对象的内存空间，**操作对象实际上是在操作对象的引用**，而不是实际的对象，因此引用类型的值是按引用访问的。

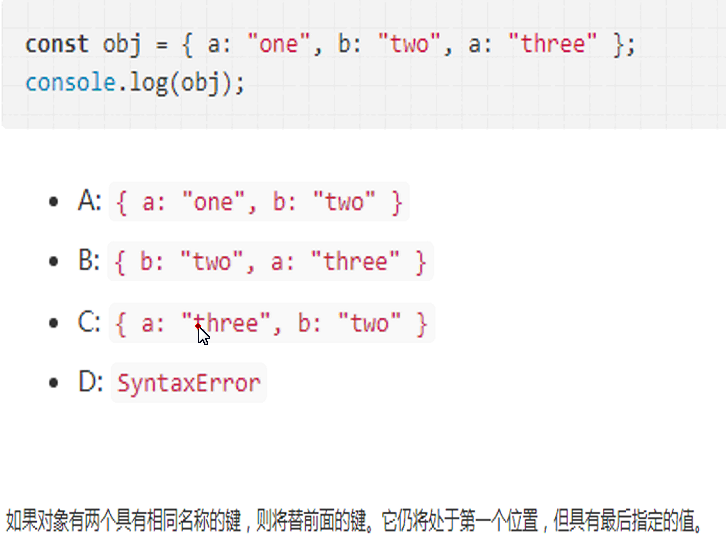
当复制保存着对象的某个变量时，操作的是对象的引用，但在为对象添加属性时，操作的是实际的对象。**只能给引用类型动态的添加属性**

引用传递是将一个变量内保存的地址传递给另一个变量



**但也只对第一个地址操作有效**

访问对象或数组之中的某个下标key的时候，如果该下标key值没有声明，没有赋值，此时取得的值是undefined类型，即便没有声明也不是引用错误



### 数组

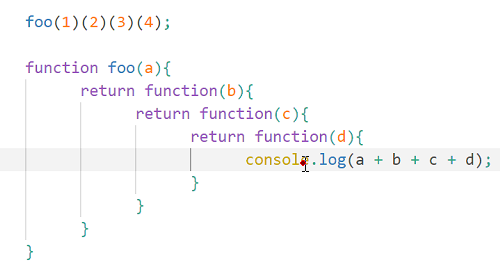
**数组的声明方式**

Var arr = [] 创建一个名为arr的空数组

Var arr = new Array(10); 创建一个名为arr长度为10的空数组，数组目前有10个undefined

创建数组的时候可以同时存放不同数据类型的数值

读取代码时应该从左向右分开读取，因为解析器是从左向右读取并执行代码的



**清空数组的方式**

Arr = 0

Arr = 【】

数组中项的修改 ： **静态赋值** arr[0] = 10；

**数组API，专门为数组类型创建的API，仅对数组有效**

**Array.prototype.concat()**方法

用于合并两个或多个数组。此方法不会更改现有数组，而是**返回一个新的数组**

**Array.prototype.join()**方法

将一个数组会一个类数组对象的所有元素连接成一个字符串。如果数组只有一个项目，那么将返回该项目不适用分隔符，**该方法会返回一个新的字符串**

var elements = ['Fire', 'Air', 'Water'];

console.log(elements.join());

console.log(elements.join(''));

console.log(elements.join('-'));



**Array.prototype.pop()方法**

删除数组中的最后一个元素，并将该元素返回

**Array.prototype.push()**

在数组的最后边添加新元素，并返回数组的长度

**Array.prototype.indexOf()**

返回在数组中找到一个给定元素的第一个索引，如果没找到返回-1

**Array.prototype.lastindexOf()**

返回在数组中找到一个给定元素的索引，从后向前找，如果没找到返回-1

**Array.prototype.shift()**

删除数组中的第一个元素，并返回该元素的值，如果数组为空返回undefined

**Array.prototype.unshift()**

将一个或多个元素添加到数组的开头，并返回该数组的长度

**Array.prototype.slice()**

返回一个新的数组对象，这一对象是一个begin和end（不包括end）决定的原数组的浅拷贝。**原始数组不会改变**

var animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];

console.log(animals.slice(2));

// expected output: Array ["camel", "duck", "elephant"]

console.log(animals.slice(2, 4));

// expected output: Array ["camel", "duck"]

console.log(animals.slice(1, 5));

// expected output: Array ["bison", "camel", "duck", "elephant"]

**Array.prototype.splice()**

通过删除或替换现有元素或者原地添加新的元素来修改数组，并以数组形式返回被修改的内容。**此方法会改变原数组**。

var months = ['Jan', 'March', 'April', 'June'];

months.splice(1, 0, 'Feb');

// inserts at index 1

console.log(months);

// expected output: Array ['Jan', 'Feb', 'March', 'April', 'June']

months.splice(4, 1, 'May');

// replaces 1 element at index 4

console.log(months);

// expected output: Array ['Jan', 'Feb', 'March', 'April', 'May']

**Array.prototype.sort()**

该方法用原地算法对数组的元素进行排序，并返回数组。默认排序是根据字符串的Unicode码点。由于它取决于具体实现，因此无法保证排序的时间和空间复杂性。

**Array.prototypr.toString()**

返回一个字符串，表示指定的数组及元素。

var array1 = [1, 2, 'a', '1a'];

console.log(array1.toString());

// expected output: "1,2,a,1a"

**数组排序和去重**

桶排序 用极致的空间复杂度降低时间复杂度，如果有重复的数桶排序自动去除，即不允许有两个重复的数

for in方式得到的结果一定是字符串

快速排序：找中点，分左右

去重方式：1 声明新数组，利用indexof方法去重

2 利用桶排序去重

3



Set结构是不允许重复的结构

slice（）方法可以实现伪数组转化为真数组

数组类的方法只有数组才能调用

js中this是一个方法的调用者

伪数组转化为真数组,就可以调用数组中的很多方法

1. 自己写代码实现



1. 自己封装slice，得到slice原生代码再返回出去

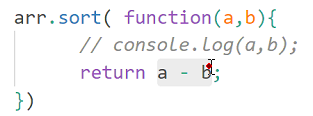




**Array.Sort（）方法**

数组中的sort只会用第一个字符进行排序，sort的应用

利用sort按从小到大的顺序排序



利用sort（）函数按从大到小的顺序进行排序

arr.sort(function(a, b){

return b - a;

})

## 第二周

### 字符串

字符串和数组的区别：数组可以存任意的数据类型，但是字符串只能存字符

**字符串也可以像数组一样用下标去除某个字符**

console.log(str.charAt(0));和console.log(str[0])取出某一字符的结果一样

**字符串API**

**String.replace();**

字符串的replace会创建一个新的字符串，不会改变原来的字符串，原来的字符串不会做替换，str.replace(str1, str2)或str.replace(/a|b/g, “\*”)前一个表示要查找的内容，后一个表示替换的内容

**String.repeat();**

.repeat()函数表示重复的字符串，参数表示重复的次数

**String.concat();**

**String.indexOf();**

返回字符串参数第一个索引对应下标位置，若没有则返回-1；

**String.lastindexOf()**

返回字符串从后往前查第一个索引对应下标，若没有则返回-1；

**String.split();**

根据参数来对字符串进行分割并返回一个新的数组类型

**String.substring();**

**String.sclice();**

**String.fromCharCode();**

**String.charCodeAt();**

**String.trim();**

**String.padEnd();**

**String.padStart();**

**JSON**

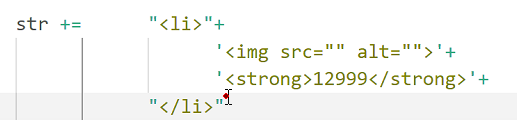
Json格式是高效的数据储存格式，JSON中一定存的是字符串，JSON就是对象！

Var JSON = {“key”：“value”}，在对象中应该是以键值对的形式存放数据，value中应该是数组形式，数组内存放不同的个体对象

将json格式的转化为html进行页面渲染

以数据为导向的页面渲染

Json标准格式中key值和value值都是字符串



在ES5中，不能识别换行，硬分开拼接

JSON中的数据格式：

第一级结构最好是对象

第二级结构根据数据数量决定，一般都是数组

第三级结构 描述个体，一般都是对象

使用常量定义字符串和使用new关键字定义字符的区别

var str1 = new String("abc");

使用new关键字表示创建一个对象， object类型

var str2 = "abc";

使用常量定义表示创建一个字符串， string类型

str1 === str2 false

str1 === str2 true

json本身是字符串，是后端给前端穿的字符串，传递数据效率最高

**ASCII**

Acsii码 数字范围：48-57

大写字母范围：65-90

小写字母范围：97-122

**Math**

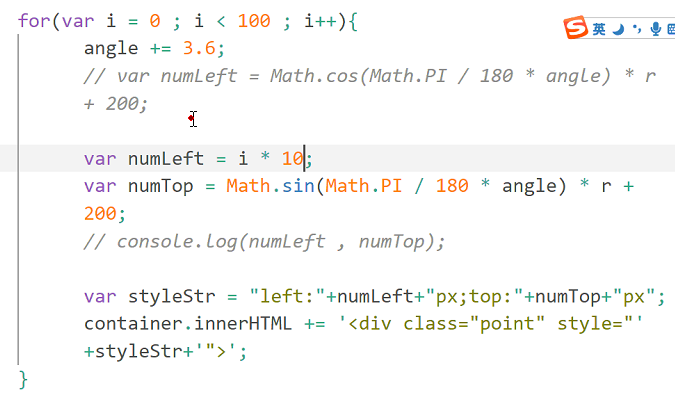
常见API

Math.random() Math.round() Math.max() Math.min() Math.abs()

Math.pow() Math.aqrt() Math.floor() Math.ceil()

三角函数

Math.sin() Math.cos() Math.tan()



**Date**

时间戳 起源：1970 1 1，之前的时间戳都是负数

Date实例对month的操作无论取直还是赋值，都是从0开始的

声明方法

直接声明 var d =new Date()

传参 var d =new Date("2015/08/12")

常用API

getYear()/getFullyear()；getMonth()；getDate()；getHours()；getMinutes()；getSeconds()

getMilliseconds()得到毫秒数

setYear()/setFullyear()；setMonth()；setDate()；setHours；setMinutes；setSeconds

setMilliseconds()设置毫秒数

Date.now()获取当前时间戳，毫秒表示；d.getTime()

toDateString()返回日期字符串

**定时器**

**开启定时器：Var id = setInterval(function,1000)**

浏览器每隔一秒执行一次function，该方法会返回一个默认数值，从1开始

调用该方法时，可以设置一个变量来存储返回值，方便终止

function无法直接传参，有两种传参方式

setInterval("function(a,b)",1000)用双引号括起来

函数设置为没有返回值

**关闭clearInterval(id)**

**延时器**

**setTimeout(function,1000)**



### BOM

#### Window

Window是BOM的对象，是绝对的最顶层的对象，优先级最高，我们把所有的全局对象放在window对象储存

全局变量和伪全局变量的区别：全局变量的保护机制：不可被删除，伪全局变量可以被删除

**window把浏览器的功能划分为很多子对象**

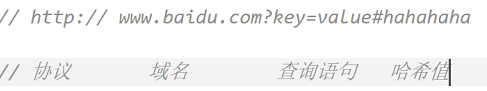
**history方法**





go正值前进，负值后退

**Location方法:地址栏有限最多容纳4KB**

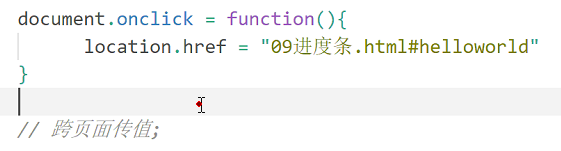




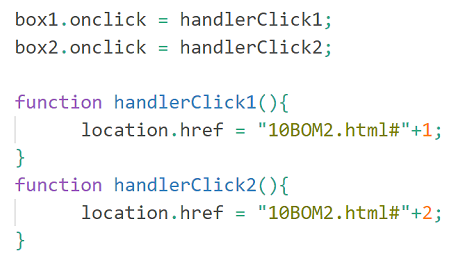
href在浏览器中不可存在中文，会被转码为unicode编码，所以文件名不应该写中文



**Location.hash进行跨页面传值**



传入hash值



在另一个界面获得传入的hash值 从1之后的所有值



**Location.reload()**

**重载 即刷新 执行到这句代码页面直接进行刷新**

**提示框**

alert()警告框；confirm提示框，确认返回1，取消返回0；prompt输入框，同上

**eval()**

**作用**

可以将字符串变成js代码执行

可以把json字符串转成对象

eval会把问题放在浏览器source的vm中，不会直接报错具体位置，即在引擎中报错，而不再html中解析，所以eval不要放入复杂代码

若按对象类型解析需要加()运算，不然就根据js代码进行解析

在页面上直接输入 “alert(“hello”)” js解析器不会进行字符串中代码的解析，可以通过eval（）函数进行解析

eval（“alert(“hello”)”）会执行alert（）函数

**this**

相当于指针，仅限于执行环境中存在

在页面初始化时，最先执行window执行环境，window其实是一个永久的执行环境

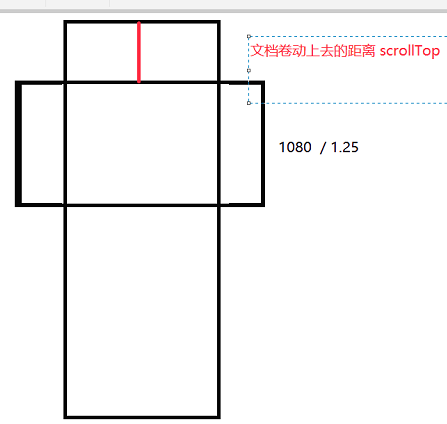
建议this使用在对象中

当事件发生时，我们事件处理函数的this会指向当前发生事件的元素

setInterval函数由浏览器执行，故指向window



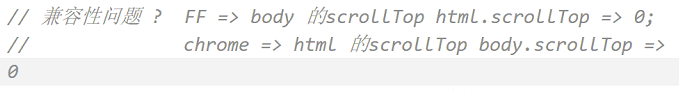
this具有就近原则，指向只看最近的函数

**案例**





兼容性问题



清除原来的所有样式

### DOM

dom是页面显示的第一承载者，吧一个标签解析成一个对象dom

dom是js最有用的对象，但也是js设计最失败的对象，因为dom是一个复杂庞大消耗性能的很重要的对象

**dom结构**

document为源，html为根节点/root，

查找元素时可以使用选择器，原生js选择器性能很差，id选择器性能最好

对dom对象的遍历，jQuery是高性能的选择器

**Dom选择器（查询元素）**

1. id选择器：getElementById（）范围+选择器

document.getElementById(“box”)

这种方法是找到了引用而不是创建了对象

1. 标签选择器

var divs = document.getElementsByIagName(“div”)

获得的是伪数组HTMLCollection

所有多个元素选择器，获取来的元素是伪数组 HTMLCollection和NodeList类型的，要用数组的方法时需要转化为数组

id选择器：单个选择器

tag选择器：一组 node或伪数组

name选择器：name在表单里代替id选择器（现在不经常使用,）

console.log(document.getElementByName(“box”));

class选择器 document.getElementByClassName(“box”);

class 选择器性能极差，兼容性不好，IE8+不建议使用

query选择器只能兼容IE8+除了兼容性不好，其他方面性能都比较好

document.querySelector() //选择单个元素的

document.querySelectorAll() //选择多个重复元素的

**增加元素**

1.增加操作

document.createElement(“div”)

2.将产生的元素放入页面之中使用appendChild（）方法

表示向后插入 插入父元素的最后一个

增加操作案例

var div = document.createElement(“div”); //增加一个元素

div.innerHTML = “helloworld”; //元素中加入内容

document.body.appendChild(div); //将增加的元素放入页面之中

div在js中创建之后要放入到页面，否则只是存在内存中，没有渲染到页面上

页面上的所有东西都是dom

如果appendChild（）方法要操作的元素在页面上存在，appendChild会做两件事

1. 删除页面上的元素
2. 重新对元素进行插入

insertBefore（）需要两个参数，1、newnode 2、oldnode

第一个参数表示新创建的节点，第二个参数表示在哪个元素前边那

class名获得所有的节点的HTMLCollection的伪数组，存有所有class同名的节点

通过id名获得的是单个的节点

**js中的children属性代表document.body里面的所有的子集元素**，

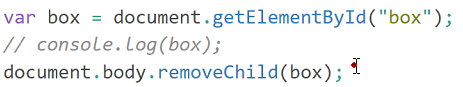
document.body.children

如果要插入到元素的最前方，应该找到父元素中所有子元素的第一个，插入到第一个之前

document.body.insertBefore(div,document.body.children[0])

**删除元素**

原来兼容性良好的写法



简化写法

找到某个节点直接删除，可以直接使用box.remove()方法删除任意节点

**修改元素**

**修改元素页面内容**：innerHTML、innerText

box.innerHTML 和box.innerText 之间有什么区别

在写入字符串时，box.innerHTML不做转义，直接解析，可以解析标签

box.innerText先将字符串进行转义，不对标签进行解析，输入啥就输出啥

**修改元素属性**

setAttribute(); getAttribute(); removeAttribute();

box.setAttribute(“hello”, “world”);

box.removeAttribute(“id”);

box.getAttribute(“class”);

为了删除掉某些存在于元素上的行内样式，利用box.style.cssText非常不优雅

应该把属性放在class里面，js只操作元素的class增删，不对那么多的属性进行同时的增删改查，会把样式放进行为之中

**js中对属性的操作简写方法**

box.id = “hello”;

box.className = “hello”;

box.title = “hello”

对属性简写时，class名都写为className ，class是js中的关键字，不可直接简写

**元素属性值的获取**

img.style.width = “300px”;

像这种dom.style.attr形式的赋值可以，但是取值尽量不要用，因为只能获取行内样式

赋值的方式：（对性能消耗及其严重）

**Offset系列方法**

offset方法得到的宽高都是盒模型的宽高，而不是内部的内容的宽高

浏览器处理offset方法时先从文档流拿出来测量后放回去，所以会造成回流，应该尽量避免使用

img.onload = function(){

console.log(img.offsetWidth) //这种获得元素宽高的方法会让页面出现回流

}

准确获取css内部宽度的方法：

**getComputedStyle（）方法**

获得的是元素的实际的宽高，即元素中内容区的宽高，但是非常消耗性能，getComputedStyle（）方法可以获取完整的CSS属性列表

img.onload = function(){

console.log(getComputedStyle(img)[“width”]);

}

js中如果要设置元素的属性margin-left时不能带 - ，因为 - 在js中会被解析为减号，所有的带 - 号的在js中都会被改写为去掉 - 符号然后后面的单词首字母大写

dom.style.marginLeft = “200px”

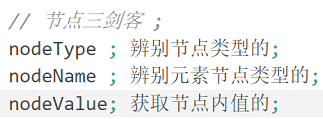
自动滚动到可视区

**节点**

节点是dom对象的统称，节点将dom进行了更细致的划分

节点的分类更细致

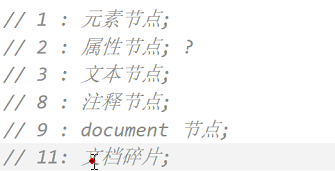
1. 元素节点
2. 文本节点
3. 属性节点



**Box.children 找到所有的子元素**

**Box.childNodes 找到所有的子节点**

nodeType其实是节点类型的代码



**文本节点包括空格 换行 tab等**

节点都可以赋值，元素节点都可以有属性节点

**属性节点是元素节点的子集，不能独立存在，只能依附元素节点存在**

属性节点的赋值

页面中所有可以看见的东西都是节点，这些节点普遍都可以创建，以对象形式存在

**文档碎片**

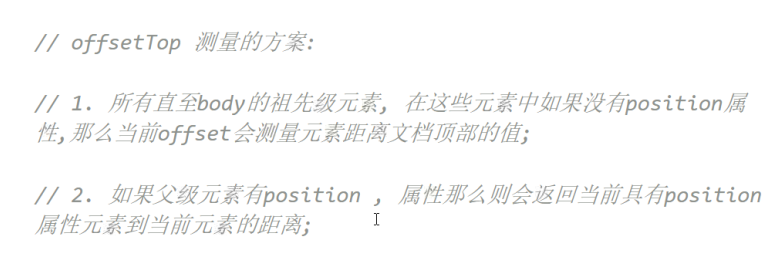


**offsetTop测量方案**

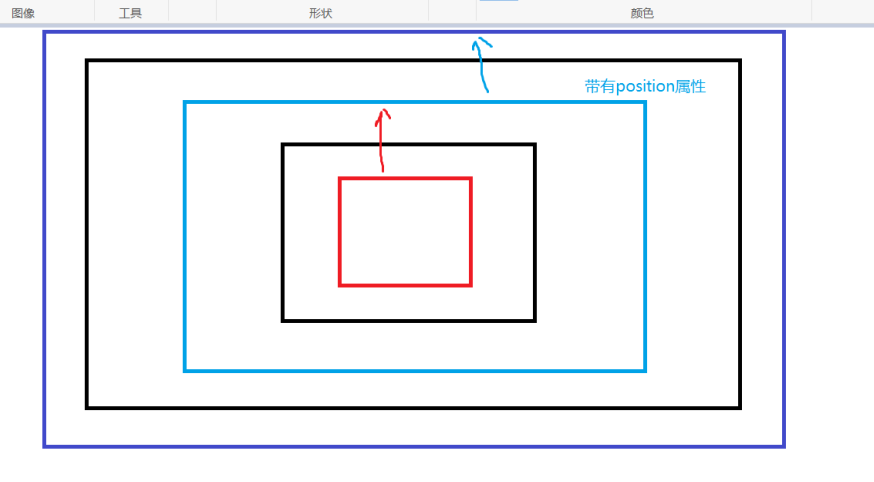
1、所有直至body的祖先级元素，在这些元素中如果没有position属性，那么当前offset

会测量元素距离文档顶部的值

2、如果父集元素有position



Box.Offsetparent返回根据哪个父元素的标签



### 事件event

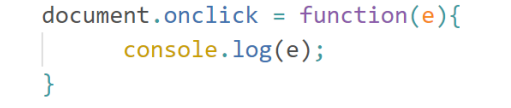
事件：用户触发的浏览器交互操作

对象：结构化储存数据

事件对象：用于储存事件发生时所有信息的对象

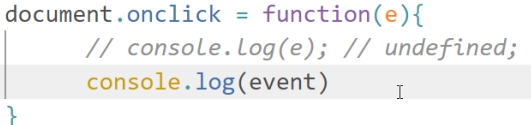
事件对象分成几种形式存在 也就是说拿到的引用方式不一样

1. 浏览器调用时，直接给事件处理函数传递一个参数

获取鼠标事件对象



火狐获取方式：



1. 浏览器执行事件时，会给全局的event对象复制当前事件的参数

推荐写法



如果我们处理取值类的兼容性问题，通常使用或(||)运算实现全部功能。

事件对象里面储存的数据

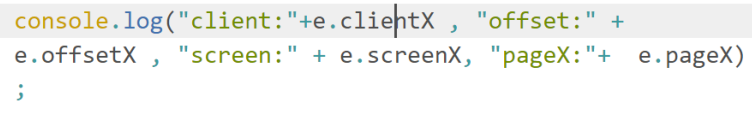


鼠标按键 左中右的顺序button 0 1 2 ； buttons 1 4 2



client offset screen page 四个属性参照物不同

可视窗口、当前区域、屏幕、文档





左上角的圈圈代表起始点

Client 当前点击地方到浏览器窗口的距离

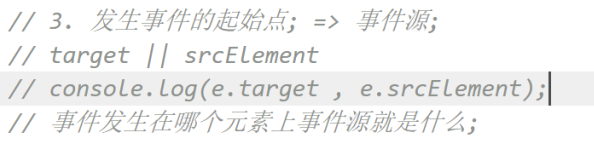
Offset 当前点击元素到自己边框的距离

Screen 当前点击地方到整个屏幕的距离

Page =>client +scrollleft 当前点击地方到html文档边缘的距离



逻辑上的根节点是Html，dom不存在于页面上

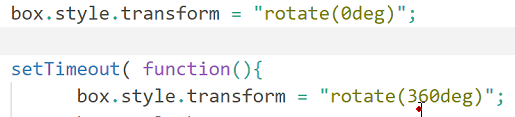


timeStamp时间戳，标志事件的独一无二



Html的可视区高度，document.documentElem获取可视区的高度

 c3中的动画效果

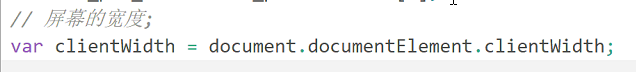


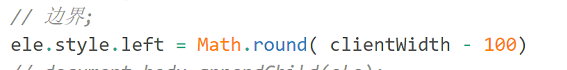
转圈 要设置一个初始值



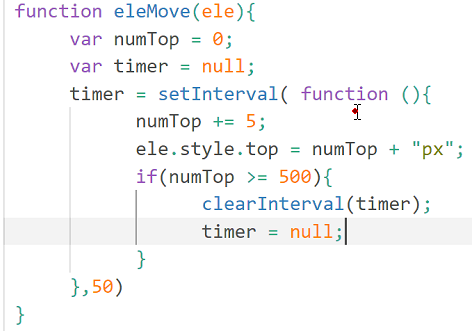


兼容性问题









css3中实现页面动画效果：transition：all 1s

transform：rotate(180deg)

实现动画效果的元素应该设置初始值：rotate（0deg）