第一章和第二章总结：

JavaScript 的发展历史： 1995年，只是为了做一个验证，之后发现向服务器发送请求前可以做出详细的验证要求，可以大大减轻服务器的压力，这样就开始发展JavaScript这个脚本语言，并迅速的成长为现代浏览器的不可获取的脚本语言，增加了与后台交互和渲染页面等强大的功能，主要分为三个方面： 1，ECMAScript 规定的基本语法和规定 ；2，DOM1 修改页面中的节点 ； 3 ，DOM2 鼠标事件和点击事件等

用script标签引入js的方式用两种： 一是直接在标签内部写js代码，二是用src属性来引入外部js文件。 ； script 标签内type属性一般都是text/javascript ,若script中没有type属性时，默认的也是text/javascript ,要注意的是引入外部文件的js时，script标签内部写的js代码是无效的； asaync： 异步加载属性，只对外部脚本有效，表示立即下载脚本，但不妨碍页面的其他操作； defer:脚本延迟属性，用来延迟脚本的执行时间，知道HTML文档已经全部解析和显示之后再执行，只对外部脚本文件有效； 平时咱们最好使用src属性来引入外部js 文件，原因： 1.可维护性高，2可以统一做修改，3，利于其他属性的添加，如defer属性等

第三章总结：

ECAMScript 的规范如：

标识符：只能是字母、下划线或$符号开头，不能使关键字

JS的基本数据类型： undefined NAN 字符串 数字 布尔值

Undefined： 声明变量但是没有初始化。

NAN: 数字计算错误

字符串： toString() String(params)

数字： Number parseInt parseFloat 计算超过计算机的范围是最大值： Infinity ;

最小值： -Infinity;

布尔值： true false

操作符： typeof也属于操作符

一位操作符如： ++i ;

加：+； 减： -； 求余： %；

基本语法： if..else do..while for..in(列举出对象的属性) for

逻辑运算符： && 与； || 或；

函数： return 返回值 ； 函数中括号中的参数； arguments.length为函数实参个数，arguments.callee引用函数自身

虽然arguments对象并不是一个数组，但是访问单个参数的方式与访问数组元素的方式相同

第四章 总结

通常的变量基本都是通过数值赋值，若变量为对象时，是应用的方式，它们应用的都是同一个堆中的对象

变量为数值是固定的大小，存放到内存中的栈中，而变量为对象时，存放到堆中。

函数都是通过数值来传递参数的

函数内部如果是对象的话也是通过数值来传递的，而不是引用传递

作用域链：函数内部局部变量在函数内部没有找到的情况下，会逐级向上查找，如果没有全局也没有找到报错

只有C和C++ 语言才需要去手动启动垃圾回收机制，而JavaScript 无需手动清除，隔一段时间会自动启动垃圾回收机制。

垃圾回收机制: 标记清除： 先标记所有，用到的去除标记，之后吧所有带有标记的都清除，来解放内存。

引用计数： 引用到的就计数，没有引用到的不计数，之后清除计数为0的对象，这种清除内存的方式有一个弊端就是对象之间的属性相互引用时，就成了循环引用，永远不会被清除，

当然，有时候咱们自己可以手动来清除对象，让其作为null,之后若是垃圾回收机制启动就会把这部分置空的对象清除，来释放内存。

第五章 引用类型

创建有两种方式：一种是 new 出一个新的对象，另一种是字面量方式创建对象

对象中取值有两种方式： 一种是点的方式取值，另一种是中括号方式取值，中括号中是字符串；两者的区别是中括号的方式可以去变量的值，而点的方式不可以。

数组拼接：

concat(): 把一个数组和另一个数组拼接在一起，返回拼接好的数组；

join(): 把数组中的每一项 按照指定的分隔符拼接成字符串；

排序：

reverse(): 倒序数组 返回值倒序数组 原有数组改变

sort(): 匿名函数进行冒泡排序 b-a倒序 a-b升序

Array 类型：

push() ： 往数组最后一位添加；

pop(): 去除数组中的最后一位；

shift(): 去除数组中的第一个；

unshift(): 往数组中第一位添加；

slice(): 若只有个数字时就是从这个数字的下标截取到结束，若是两个数字时，从开始数字的下标到结束数字的下标，（注：截取的数字不包括结束数组的下标）；

splice(): 若是一个数字的时候，表示从这个数字的下标删除到最后；有两个数字，表示第一个数是开始下标，第二个数是删除几个； 若第二个数字后还有就是往里面添加的这些；

indexOf(): 返回获取项在数组中的索引；

lastIndexOf: 返回获取到数组中出现的最后一次索引；

reduce(): 从数组的第一项开始逐个遍历到最后；

reduceRight(): 从数组的最后一项开始，向前遍历到第一项；

reduce()和reduceRight的函数接受4个参数： 前一个值、当前值、项的索引和数组对象。这个函数返回的任何值都会作为第一个参数自动传给下一项。第一次迭代发生在数组的第二项，因此第一个参数是数组的第一项，第二个参数就是数组的第二项。

Date 类型

Date 有两个方法 Date.parse()和Date.UTC() ；一般我们不需要写会自动转换

如：var person = new Date(2000)和var person = new Date(Date.parse(2000)) 效果一样。

时间的中间差的计算：

//开始

var start = new Date();

//调用函数

doSomething();

//结束

var end = new Date();

时间差： end – start

Date 有很多方法来获取或设置日期，如： getTime(),setTime(), gethours(),sethours()等还有很多；

RegExp类型

g : 表全局 i: 不区分大小写 m: 多行模式

元字符都必须转义，正则表达式中的元字符包括：

( [ { \ ^ $ | } ] ) ? \* + .

RegExp实例方法：

var text = “ mom and dad and bady”;

var pattern = /mom(and dad(and bady)?)?/gi;

var matches = pattern.exec(text);

包含两个额外的属性： index：表示匹配项中在字符串中的位置； input：表示应用正则表达式的字符串。

alert(matches.index); // 0

alert(matches.input); //”mom and dad and bady”

alert(matches[0]); // “ mom and dad and bady”

alert(matches[2]); // “ and bady ”

Function 类型

每一个函数都是由Function 来创建的实例；

函数没有重载，若前面已经定义了一个函数，后面定义的函数与前面的相同，就会覆盖之前的函数；

函数表达式和函数声明的不同之处：

var num = function(){} 函数表达式 ： 在JS运行时确定，并且在表达式赋值完成后，该函数才能调用。

function sum(){} 函数声明： 函数声明在JS解析时进行函数提升，因此在同一个作用域内，不管函数声明在哪里定义，该函数都可以进行调用。

函数中的参数，可以的一个或多个，可以是函数；

函数内部，有两个特殊的对象：arguments和 this :

arguments的主要用途是保存函数参数，但这个对象还有一个名叫callee的属性，该属性是一个指针，指向拥有这个arguments对象的函数。如：

function factorial (num){

if(num<=1){

return 1;

}else{

Return num \* arguments.callee(num-1);

}

}

函数属性和方法

函数是对象，因此函数也有属性和方法。每个函数都包含两个属性： length 和 prototype.其中，length属性表示函数希望接受的命名参数的个数。如：

function sum(num1,num2){

return num1 + num2;

}

alert(sum.length); //2

prototype 是保存它们所有实例的方法的真正所在；换句话说，诸如toString()和valueOf()等方法实际上都保存在prototype名下，只不过是通过各自对象的实例访问的；prototype属性是不可枚举的，因此使用for-in 无法发现。

每个函数都包含两个非继承而来的方法： apply() 和 call():这两个方法的用途都是在特定的作用域中调用函数，实际上等于设置函数体内this对象的值。

apply() 方法接受两个参数：一个是在其中运行函数的作用域，另一个是参数数组。其中参数可以是Array的实例，也可以是arguments对象 如：

function sum(num1,num2){

return num1 + num2;

}

function callSum1(num1,num2){

return sum.apply(this,arguments); //传入arguments对象

}

function callSum2(num1,num2){

return sum.apply(this,[num1,num2]); //传入数值

}

alert(callSum1(10,10)); //20

alert(callSum2(10,10)); //20

在上面例子中，callSum1() 在执行sum()函数时传入this作为this值（因为是在全局作用域中调用的，所以传入的就是window对象）和arguments 对象。

call()方法，第一个参数是this没有变化，变化的是其余参数都直接传递给函数，换句话说，在使用call（）方法时，传递给函数的参数必须逐个列举出来。

事实上，传递参数并非apply（）和call()真正的用武之地，他们真正强大地方是能够使能扩充函数赖以运行的作用域。如：

Window.color = “red”;

var o = { color: “blue” };

function sayColor(){

alert(this.color);

}

sayColor() ; // red;

sayColor.call(this); //red

sayColor.call(window);//red

sayColor.call(o); //blue

ES5还定义了一个方法： bind() 这个方法会创建一个函数的实例，其this值会被绑定到传给bind() 函数的值 如：

Window.color = “red”;

var o = { color: “blue” };

function sayColor(){

alert(this.color);

}

var objectSayColor = sayColor.bind(o);

objectColor(); //blue;

基本数据类型

Boolean、Null、Undefined、Number、String

用typeof来判断类型，用instanceof来判断是不是如:String, 创建的实例;

Number中的toFixed（）方法可以吧Number四舍五入为指定小数位数的数字；

String和其他Number、Boolean 类型一样，一般也是有构造函数创造，不过创造之后就瞬间销毁了，过程如下： 一般我们都不用，因为这样就很难区分引用类型和基本数据类型

var text = new String(“txt time”);

text = null;

String 中的方法有 concat()： 用于拼接字符串 里面可以传一个或多个参数；

slice(): 用于截取一段字符串，但是不会影响原来的字符串 里面也可以传一个或多个参数；

substring():用于截取一段字符串，但是不会影响原来的字符串 里面也可以传一个或多个参数；

substr():用于截取一段字符串，但是不会影响原来的字符串 里面也可以传一个或多个参数；

charAt():方法可返回指定位置的字符；

chartCodeAt(): 方法返回指定位置的字符串的Unicode编码；

indexOf(): 方法可返回某个指定的字符串值在字符串中首次出现的位置；

lastIndexOf(): 方法可返回一个指定的字符串值最后出现的位置，在一个字符串中的指定位置从后向前搜索。

indexOf()、lastIndexOf()这两个字符串的方法都对大小写敏感，若要检索的字符串没有出现，则改方法返回-1；

split(): 方法用于吧一个字符串分割成字符串数组；若吧空字符串（””）用作separator，那么stringObject中的每个字符之间都会被分割，String.split执行的操作与Array.join执行的操作是相反的。Array.join如下：

arrayObject.join(separator)

join()方法用于吧数组中的所有元素放入一个字符串。元素是通过指定的分隔符进行分割。

separator:可选。指定要使用的分隔符，若省略该参数，则使用逗号作为分隔符。

var arr = new Array(3)

arr[0] = “George”

arr[1] = “John”

arr[2] = “Thomas”

document.write(arr.join(“.”)) // George.John.Thomas

eval() 函数可计算某个字符串，并执行其中的JavaScript代码； eval（string）如：

eval(“alert(‘mom is good’)”) //mom is good

eval(“2+3”) // 5

注意：虽然eval()的功能非常强大，但在实际使用中用到它的情况并不多

特别的两个内置对象： Global属性 和 Math 属性

Global和window的区别：

global是JavaScript运行时所在宿主环境提供的全局对象

window对象时浏览器的一个web api，可以说是global在浏览器中的具体表现

global对象是单体内置对象，既不依赖宿主环境的对象，而window对象依赖浏览器。

Math如： Math.max() Math.min() 等还有正弦，余弦等等，看需要。

第六章 对象

就JavaScript而言，创建对象的方式有两种一种是 new Person（） ，另一种是字面量的方式创建

修改对象的默认属性definePrototype 包含两个参数，第一个是要修改的属性，第二个是要增加或者修改的属性

俗称可是拓展对象作用域的方法有apply(),call(),bind()

apply方法和call方法有些相似，它也是改变this的指向，同样apply也可以有多个参数，但是不同的是，第二个参数必须是一个数组，注意： 如果call和apply的第一个参数写的是null,那么this指向的是window对象。

bind方法和call、apply方法有些不同，但是不管怎么说它们都可以用来改变this的指向；

先说说它们的不同吧； 如下：

var a = {

user: “追妹子”，

fn: function (){

console.log(this.user);

}

}

var b = a.fn;

b.bind(a);

我们发现代码代码没有被打印，对，这就是bind和call、apply方法的不同，实际上bind方法返回的是一个修改过后的函数。

var a = {

user: “追妹子”，

fn: function(){

console.log(this.user)

}

}

var b = a.fn;

var c = b.bind(a);

console.log(c); //function(){ [native code] }

那么我们现在执行以下函数c看看，能不能打印出对象a里面的user

var a = {

user: “追妹子”，

fn: function(){

console.log(this.user); //追妹子

}

}

var b = a.fn;

var c = b.bind(a);

c();

同样bind也可以有多个参数，并且参数可以执行的时候再次添加，但是要注意的是，参数是按照形参的顺序进行的

var a = {

user: “追妹子”，

fn: function(e,d,f){

console.log(this.user); //追妹子

console.log(e,d,f); //10 1 2

}

}

var b = a.fn;

var c = b.bind(a,10);

c(1,2);

call和apply都是改变上下文中的this并立即执行这个函数，bind方法可以让对应的函数想什么时候调用就什么时候调用，并且可以将参数在执行的时候添加，这是它们的区别，根据自己的实际情况来选择使用。