1. **JavaScript包括那几个部分？**

JavaScript是一种专为与网页交互而设计的脚本语言，包含三个部分：

1. ECMAScript，由ECMA-262定义，提供核心语言功能；
2. 文档对象模型（DOM），提供访问和操作网页内容的方法和接口；
3. 浏览器对象模型（BOM），提供与浏览器交互的方法和接口。
4. **<script></script>标签的位置**

放在<head>元素中，与css文件一样，也就是只有在所有的css和js文件都下载、解析、执行完成之后，再开始呈现页面的内容（浏览器在遇到<body>标签才开始呈现内容），对于js代码很多的页面来说，导致浏览器在呈现页面时出现明显的延迟，延迟期间浏览器窗口一片空白。

因此一般放在</body>标签之前，即在解析js代码之前，页面内容已经呈现在浏览器中。

1. **js延迟加载的方式有哪些？**

defer和async、动态创建DOM方式（用得最多）、按需异步载入js

defer属性：将脚本延迟到浏览器遇到</html>标签后再执行，当有多个脚本文件设置defer属性时，按照顺序执行，但最好只设置一个defer属性的脚本，defer只适用于外部脚本。把延迟脚本放在页面底部最佳。在一些支持html5的浏览器会忽略defer属性。

async属性：指定async属性是为了不让页面等待两个脚本下载和执行，从而异步加载页面其他内容，因此建议异步脚本不要在加载期间修改DOM。只适用于外部脚本，标记为async的脚本并不保证按照指定的先后顺序执行。

1. **<script>标签在XHTML中的用法**

XHTML：可扩展超文本标记语言，将HTML作为XML的应用而定义的一个标准。比HTML严格。针对不兼容XHTML和交融XHTML的浏览器，把脚本放在XHTML中的兼容写法是：

<script>

//<![CDATA[

....... /\*代码部分\*/

//]]>

</script>

这种格式在所有现代浏览器中都可以用，能通过XHTML验证，而且对XHTML之前的浏览器也会平稳退化。

**5、<script>脚本最好引用外部文件引入，why?**

1. 可维护性：遍及不同HTML页面的js脚本会造成维护问题。引入外部脚本，可以在不触及HTML标记的情况下，集中精力编辑js代码。
2. 可缓存：浏览器能够根据具体的设置缓存链接的所有外部脚本文件。也就是，如果两个页面都使用同一个文件，那这个文件只需要下载一次。加快页面的加载速度。
3. 适应未来：通过外部文件来包含js无须使用前面提到XHTML/注释hack，因为HTML和XHTML包含外部文件的语法是相同的。
4. **标识符的命名规则**

第一个字符可以是字母、\_、$;

其他字符可以是字母、\_、$、数字；

不能把关键字和保留字，true，false，null用作标识符。

1. **JavaScript数据类型(6种)**

5种简单数据类型：Undefined,Null,Boolean,Number,String;

1种复杂数据类型：Object.

采用typeof判断数据类型时：未定义的值——undefined;布尔值——boolean；字符串——string;数值——number；对象/null（特殊，空对象）——object；函数——function。

Undefined：初始化但未定义的变量（var s;），typeof(s)=undefined，但是未初始化的变量在使用typeof(a)=undefined。因此显式的初始化变量是最佳选择，即var s=0;

Boolean：false情况有以下几种：””(空字符串)、0、NaN、null、undefined; 其余都是true.

Number:可以使用Number.MIN\_VALUE和Number.MAX\_VALUE来验证计算机中number的最大值和最小值。超出最大值/最小值——Infinity/-Infinity（不能参与运算）。isFinite()函数，判断是不是超出范围。NaN：NaN/10——NaN.任何数值除以非数值都得到NaN。但是NaN==NaN是false。判断是不是NaN利用isNaN()函数，函数在接收一个值时，会将其转换为数值，不能转换为数值的则返回true，其余均返回false.eg：isNaN(‘10’)—false;isNaN(true)—false,isNaN(‘blue’)—true.函数会把字符串、true等转换为数值。不是数值才返回true.

String: 如何转为字符串？toString()和String()两种方法。

**toString()方法**：数值、布尔值、对象、字符串都有这个方法，但null和undefined没有。

Var age=11;

Var ageAsString=age.toString(); //”11”

Var ageAsString1=age.toString(8); //”1011” 可以指定基数转换不同的数值格式

Var found=true;

Var foundAsString=found.toString(); //”true”

**String()方法：**适用所有类型。在未知转换的值的类型的时候可以用。

String(10) //”10” String(true) //”true” String(null) //”null”

Var s; string(s) //”undefined” s未定义

**8、JavaScript数值转换？**

3个函数：Number()/parseInt()/parseFloat()

**Number()函数**：可以用于所有数据类型。

Var num1=Number(‘hello world’); //NaN（以下简写）

字符串处理：‘hello world’ //NaN “” //0

‘123hello123world’//NaN “oxf”// 15

“01.1.123” //1.1 “0123”//123

布尔值处理：true //1 false //0

数字处理：直接传入和返回

Null：null //0

Undefined： undefined //0

**parseInt()函数**：第一个字符不是数字字符或负号——NaN

Var num1=parseInt(‘123hello world’); //123（以下简写）

“hello 123” //NaN “” //0 “oxA” //10

22.5 //22 “070”//56 “70” //70

关于八进制的问题：”070”在ES3时认为是56，ES5时认为是70，因此引入基数，即

parseInt(“070”,8) //56 parseInt(“070”,2) //70

parseInt(“AF”,16) //175

**parseFloat()函数：**会忽略前导的0,

Var num1=parseFloat(‘123hello world’); //123（以下简写）

“0xA” //0 “22.5” //22.5 “22.34.5” //22.34

“0908.54” //908.54 “3.12324e3” //3123.24

1. **JavaScript中的函数**

声明一个函数：function f1(arg1,arg2,...,argN){statements}

也可以不显式的使用命名参数：function f1(){statements},即不需要函数签名，通过访问arguments对象的length属性可以获知有多少个参数传递给了函数。

Eg: function howManyArgs(){alert(arguments.length);}

howManyArgs(“string”,45) //2

howManyArgs(1) //1

howManyArgs() //0

Arguments[0]与arg1等价，修改arguments[0]的值，arg1的值也会变化。但当不存在arguments[i]时，给其赋值，并不能改变传递的参数。

由于不存在函数签名的特性，ECMAScript函数没有重载，即两个名字相同的函数，里面执行的语句不同，第二个函数会把第一个函数覆盖。

1. **基本类型和引用类型**

基本数据类型：Number/Null/Undefined/String/Boolean，按值访问的，即可以操作保存在变量中的实际的值。

引用类型：Object 按引用访问的。

eg: 复制变量

|  |  |
| --- | --- |
|  |  |
| num2 | 5(Number类型) |
| num1 | 5(Number类型) |

Var num1=5; var num2=num1;

|  |  |
| --- | --- |
|  |  |
|  |  |
| num1 | 5(Number类型) |

复制

Var obj1=new Object(); var obj2=obj1; obj1.name=”jack”; alert(obj2.name); //”jack”

|  |  |
| --- | --- |
|  |  |
|  |  |
| obj1 | (Object类型) |

Object

Object

|  |  |
| --- | --- |
|  |  |
| obj2 | (Object类型) |
| obj1 | (Object类型) |

Object

即两个变量引用的都是一个对象，对象保存在堆内存中。复制操作，其实是复制的指针，而这个指针指向存储在堆内存中的一个对象。

因此要理解基本类型和引用类型。

1. **ECMAScript传递参数**

所有参数都是按值传递的。在向参数传递基本类型的值时，被传递的值会被复制给一个局部变量（arguments对象中的一个元素，即命名参数）；在向参数传递引用类型的值时，会把这个值在内存中的地址复制给一个局部变量，因此这个局部变量的变化会反映在函数的外部。eg1:

function addTen(num){

num+=10;

return num;

}

var count=20;

var result=addTen(count);

alert(count); //20 没有变化

alert(result);//30 num值加10，并不会改变count值，因此result变为30，而count不变。

eg2:

function setName(obj){

obj.name=”jack”;

}

var person=new Object();

setName(person);

alert(person.name); //”jack”

因为obj和person均指向一个局部变量的地址，因此修改任意一个的值，都会跟着变化。

eg3:

function setName(obj){

obj.name=”jack”;

obj=new Object(); //新建一个局部对象，在函数使用完后即被销毁

obj.name=”kate”;

}

var person=new Object();

setName(person);

alert(person.name); //”jack”

即使在函数内部修改局部变量的值，但是原始的引用仍然保持不变，证明person是按参数传递的，当在函数内部重写obj时，这个变量引用的就是一个局部对象了，而这个局部对象会在函数执行完毕后立即被销毁。自然不会改变外部的值。

1. **检测类型typeof和instanceof**

Typeof在检测类型时，无法区分对象，究竟是什么类型的对象？

因此需要instanceof操作符来完成。语法是

Result=variable instanceof constructor

所有的对象obj，执行obj instanceof Object,返回的结果都是true。基本类型进行操作为false，因此其不是对象。

1. **作用域链和延长作用域链**

每次进入一个执行环境，都会创建一个用于搜索变量和函数的作用域链；当代码在一个环境中执行时，会创建变量对象的作用域链，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。标识符的解析是按照作用域链一级一级的搜索标识符的过程。从作用域链的前端一直搜索到window对象所在的最外围全局执行环境。

eg:

var color=”red”;

Function changeColor(){

if(color==”red”){

color=”blue”;

}else{

color=”red”;

}

}

changeColor();

alert(“color is now”+color); //red

函数的作用域链包含两个对象：自己的变量对象（定义的arguments对象）和全局环境的变量对象。可以在函数内部访问color，因为在作用域链中可以找到它。

学会写一个作用域链？以及对执行环境的理解。代码可以解释。

eg:

var color=”blue”;

function changeColor(){

var anotherColor=”red”;

function swapColor(){

var tempColor=anotherColor;

anotherColor=color;

color=tempColor; //可以访问color,anotherColor,tempColor

}

swapColor(); //可以访问color,anotherColor

}

changeColor(); //可以访问color

延长作用域链的方法：

1. try-catch语句中的catch块；
2. With语句。
3. **JavaScript的垃圾回收**

JavaScript是具有自动垃圾收集机制的编程语言，开发人员不必关心内存分配和回收问题。离开作用域的值将被自动标记为可以回收，因此将在垃圾收集期间被删除。

“标记清除”是目前主流的垃圾清除算法，这种算法的思想是给当前不使用的值加上标记，然后再回收其内存。

另一种垃圾清除方法是“引用计数”，算法思想是跟踪记录所有值被引用的次数，JavaScript引擎目前都不再使用这种算法，但是在IE中访问原生JS对象（如dom元素时），这种算法仍然会导致问题。即当代码中出现循环引用对象时，“引用计数”算法就会出现问题。解除变量不仅有助于消除循环引用现象，而且对垃圾收集也有好处。

为了确保有效的回收内存，应该及时删除不再使用的全局对象、全局对象属性以及循环引用变量的引用。

1. **引用类型**
2. **Object类型**

创建方法2种：

方法1：new Object()方法： 方法2：对象字面量法：（常用）

var person=new Object(); var person={

person.name=”jack”; name: “jack”,

person.age=20; age:20 } //最后一句不加逗号

访问对象的方法2种：person.name 和person[“name”]

1. **Array类型**

创建数组的方法2种：

方法1：new Array()方法：

var colors=new Array();

var colors=new Array(20); //length为20的数组

var colors=new Array(“red”,”blue”,”green”);

var colors=Array(“red”,”blue”,”green”); //new也可以省略

方法2：数组字面量法：

var colors=[“red”,”blue”,”green”];

var name=[]; var valye=[1,2];

访问数组的方法：colors[0],colors[2].... 求数组的长度 colors.length;

注意：如果给定colors.length=5，则可以增加或者删除数组中的元素，即length属性不是只读的。可以利用colors(length)为数组在最后一位后面添加元素，因为数组最后一位是length-1.

**检测数组：**

Value instanceof Array来检测一个对象是不是数组。

但是存在一个问题，instanceof假定只有一个全局执行环境，但当引用多个框架的时候，就存在多个全局环境，如果想从一个框架传入数组到另外一个已经有原生创建的数组的全局执行环境的话，就会出现问题。——ES5新增Array.isArray()方法解决这个问题。

这个方法只是确定其是不是数组，不管是来自于哪个执行环境。

**数组转换方法：**

所有对象均有toLocaleString(),toString(),valueOf()方法。而toString(),valueOf(),alert均是调用的toString()，而toLocaleString()则不一定返回一样的值。实例如下：

var persom1={

toLocaleString: function(){

Return “Nikolas”;

},

toString: function(){

Return “Nikolas”;

}

};

var persom2={

toLocaleString: function(){

Return “Grigolos”;

},

toString: function(){

Return “Greg”;

}

};

var people=[person1,person2];

aert(people); //Nikolas,Greg 调用toString方法

aert(people.join(“-”); //Nikolas-Greg join方法可以改变不同的分隔符。

alert(people.toString()); //Nikolas,Greg 显示tostring的值

aert(people.to LocaleString()); //Nikolas,Grigolos 显示toLocaeString的值。

**栈方法：（LIFO先进后出）**

ES提供了两个方法push()和pop();

Push方法则在栈的末端添加参数，并返回数组的长度；pop方法从数组末尾移除一项，减少数组的长度，并返回移除的项；

**队列方法：（FIFO先进先出）**

队列在末端添加项，在列表前端删除项，用shift()和push()一对，unshift()和pop()一对来模拟队列。其中shift是移除数组第一项并返回该项，数组长度减1；unshift是前端添加元素，并返回数组的长度。

实例：

var colors=new Array(); //新建一个数组

var count=colors.push(“red”,”green”); //colors中加入两个元素

alert(“count”); //2 栈的push方法，返回数组长度；

var item=colors.pop();

alert(item); //green 栈的pop方法，返回最末端的元素，数组长度减1

alert(colors.length); //1 现在colors中只剩下red

var count2=colors.push(“black”,”yellow”); //red black,yellow

var item2=colors.shift();

alert(item2); //red 队列的shift方法返回数组的前端元素。数组长度减1；

var item3=colors.unshift(“white”); //white,black,yellow

alert(item3); //3 队列的unshift方法，在前端插入元素，并返回数组长度

**数组的重排序方法：**

reverse()和sort();

eg:

var values=[1,2,3,4,5];

values.reverse();

alert(values); //5,4,3,2,1 因此reverse方法就是反转数组项的顺序

var values1=[0,1,5,10,15];

values1.sort();

alert(values); //0,1,10,15,5 sort函数比较的是字符串，因此加入一个比较函数较好

values.sort(compare); alert(values); //0,1,5,10,15

value.reverse(); //15,10,5,1,0 在使用reverse就可以得到倒的排序了

function compare(value1,value2){ //比较函数如下所示

If(value1>value2){

return -1;

}else if(value1<value2){

return 1;

}else{

return 0;

}

}

**数组的操作方法：**

concat(),slice(),splice();

eg:

var colots=[“red”,”green”,”blue”];

Var colors2=colors.concat(“yellow”,[“black”,”brown”]);

Alert(colors); //red,green,blue 因此concat方法只是复制当前副本，并没有改变原来的数组。

Alert(colors2); //red,green,blue,yellow,black,brown

eg:

var colors=[“red”,”green”,”blue”,”yellow”,”purple”];

Var colors2=colors.slice(1); alert(colors2); //green,blue,yellow,purple

Var colors3=colors.slice(1,3); alert(colors3); //green,blue,yellow

alert(colors); //red,green,blue,yellow,purple

因此slice只是截取部分数组中的元素。且不影响原始数组。

eg：

var colots=[“red”,”green”,”blue”];

Var removed=colors.splice(0,1);

Alert(colors); //green,blue

Alert(removed); //red 从第0项开始，删除几个元素

Removed=colors.splice(1,0,”yellow”,”orange”);

Alert(removed); //空数组，因为表示从第1个开始，删除0个元素，并在此位置加入元素

Alert(“colors); // green,yellow,orange,blue

Removed=clors.splice(1,1,”red”,”purple”);

Alert(removed); //yellow，因为表示从第1个开始，删除1个元素，并在此位置加入元素

Alert(“colors); // green,red,purple,orange,blue

数组的位置方法：indexOf和lastIndexOf()

两个可选项：要查找的项（可选）+查找起点位置的索引。

Var numbers=[1,2,3,4,5,4,3,2,1];

Alert(numbers.indexOf(4)); //3 表示正序索引为4的项

Alert(numbers.lastIndexOf(4)); //4 表示倒序索引为4的项

Alert(numbers.indexOf(4,4)); //5 表示从正序索引为4开始查找4的项所在的索引。

Alert(numbers.lastIndexOf(4,4)); //3 表示从倒序索引为4开始查找4的项所在的索引。

**数组的迭代方法：**

every(),filter(),forEach(),map(),some()

用法实例：

Var numbers=[1,2,3,4,5,4,3,2,1];

Var everyResult=numbers.every(function(item,index,array){

Return intem>2;

}); alert(everyResult); //false必须得所有都满足item>2才返回true

Var someResult=numbers.every(function(item,index,array){

Return intem>2;

}); alert(someResult); //true只要有一个满足item>2就返回true

**数组的归并方法：**

reduce()和reduceRight()

Reduce表示从数组第一项遍历到最后，reduceRight表示从最后一项遍历到最前端；接收四个参数（前一个值，当前值，项的索引，数组对象）。

var values=[1,2,3,4,5];

var sum=values.reduce(function(prev,cur,index,array){

return prev+cur;

}); alert(sum); //15

1. **Date类型**

创建一个日期对象方法：var now=new Date();

不传递参数的情况下，自动获得当前日期和时间，想获得特定时间的话，Date.parse()和Date.UTC(),使用方法是var now=new Date(Date.parse(“May 25,2004”));

ar now=new Date(“May 25,2004”);两种写法是等价的，后台也会调用Date.parse()方法

var now=new Date(Date.UTC(2000,0,5,17,44,55));表示2000年1月5号下午17:44:55这里的前两项是必选的，后面可选，由于月份默认加1，因此1月的话要写为0.

1. **RgeExp类型**

ES通过RgeExp类型来表示正则表达式。

正则表达式的匹配模式支持3个标志：

g:全局模式，表示模式应用于所有字符串，而非在发现第一个匹配项时立即停止；

I：表示不区分大小写模式，即在确定匹配项时忽略模式与字符串的大小写；

m: 多行模式，在到达一行文本文末时还会继续查找下一行中是否存在模式匹配的项。

。。。。待补充

1. **Function 类型**

**ECMAScript没有重载（深入理解）：**

function addSomeNumber(num){

return num+100;

}

function addSomeNumber(num){

return num+200;

}

Var result=addSomeNumber(100); //300

第二个函数把第一个覆盖了，这就是没有重载，可以把函数名想象为指针，函数名指针对应的函数对象为函数内容，当第二个函数名修改后，也就是把第一个函数内容修改了。因此第二个看起来是覆盖了第一个函数。

**函数声明提升问题：**

alert(sum(10,10));

function sum(num1,num2){ //函数声明

return num1+num2;

}

解析器会先声明函数并把它放在顶端，然后再逐个表达式的解析，因此会出现函数声明提升问题，但是：

alert(sum(10,10)); //直接执行会出现错误。

var sum=function(num1,num2){ //不属于函数声明，函数位于初始化语句中。

return num1+num2;

}

**作为值的函数：（即函数内部嵌套函数）**

function callSomeFunction(someFunction,someArgument){

return someFunction(someArgument);

}

Eg：

function add10(num){

return num+10;

}

var result=callSomeFunction(add10,10); //函数作为值传入一个函数内部。即作为值的函数。

alert(result); //20

常用的排序函数：

Function createComparisonFunction(prototyName){

return function(object1,object2){

Var value1=object1[prototyName];

Var value2=object2[prototyName];

If(value1<value2){

return -1;

}else if(value1>value2){

return 1;

}else{

return 0;

}

};

}

排序函数用法：

var data=[{name:”Zara”,age:26},{name:”Nack”,age:25}];

data.sort(createComparisonFunction(“name”));

alert(data[0].name); //Nack

data.sort(createComparisonFunction(“age”));

alert(data[0].name); //Zara

call 和apply的区别：

1. 接收参数的方式不同，call第一个参数是this，变化的其余参数都直接传递给函数，即在使用call方法时，传递给函数的参数要一一列举出来。而apply直接可以传入arguments对象或者数组。但在不传递参数的情况下，两者都一样。
2. 两者最重要的作用是扩充函数的作用域。对象不需要与方法有任何耦合关系。（此处要深刻理解P118）
3. **基本包装类型**

为了便于操作基本类型值，ES提供了3个特殊的引用类型：Boolean/Number/String

**Boolean类型：**

创建方法：

Var a=new Boolean(true);

重写valueOf()和toString()方法：

valueOf返回true/false;toString()方法返回”true”/”false”;

布尔表达式中的所有对象都会被转换为true；下面例子区分一下：

eg:

var falseObject=new Boolean(false);

var result=falseObject&&true;

alert(result); //true

var falseObject=false;

var result=falseObject&&true;

alert(result); //false

基本类型与引用类型的布尔值的区别：

1. Typeof操作符的区别，返回值分别是boolean/object;
2. instanceof操作符：测试Boolean对象返回true，测试基本类型布尔值返回false;

**Number类型：**

Var a=new Number(10);

重写valueOf(),toString(),toLocaleString()函数：valueOf()返回对应的数值；另外两个返回对应字符串。toString()还可以传递一个基数，2,8,16等。toString(8);

toFixed()方法：按照指定的小数位返回数值的字符串表示。

var num=10;

alert(num.toFixed(2)); //”10.00”记住，是字符串

toExponential()方法：返回对应的指数表示法，可以传递一个参数，即保留的小数位。

toPrecision()方法：返回最合适的格式。

String类型：

Var s=new String(“hello world!”);

**Js重点部分（面向对象程序设计）**

**1、属性类型：**

ECMAScript有两种属性：数据属性和访问器属性。

**数据属性**包含一个数据值的位置，可以读取和写入值，4个特性：

[[Configurable]]: true（默认），可以删除并定义，false不可以删除，不能定义

[[Enumerable]]: 表示能否通过for-in循环返回属性

[[Writable]]: 能否修改属性的值

[[Value]]: 属性的数据值

**访问器属性**不包含数据值，包含一对getter，setter函数，读取访问器属性时，调用getter函数，读取调用setter。访问器属性有4个特性：

[[Configurable]]: true（默认），可以删除并定义，false不可以删除，不能定义

[[Enumerable]]: 表示能否通过for-in循环返回属性

[[Get]]: 读取属性调用的函数

[[Set]]: 写入属性时调用的函数

**修改属性**默认特性，用Object.defineProperty()方法，接收三个参数：属性所在的对象，属性的名字，一个描述符对象（configurable,enumerable，writable, value）。

**定义多个属性**的方法：Object.defineProperties()方法,接收两个参数：要添加和修改其属性的对象，对象的属性与第一个对象中要添加或修改的属性一一对应。

**读取属性**的特性：Object.getOwnPropertyDescriptor()方法,接收两个参数：属性所在的对象和要读取其描述符的属性名称。返回值是一个对象，如果是访问器属性，对象的属性有configurable,enumerable,get,set;如果是数据属性，对象的属性有configurable，enumerable,writable,value.

eg：

Var book={};

Object.defineProperties(book,{ //定义多个属性

\_year:{value:2004;}, //定义数据属性，即添加或修改对象的值

Edition:{value:1},

Year:{ //定义访问器属性，与数据属性一一对应

Get:function(){return this.\_year;}, //读取调用的函数

Set:function(newValue){ //写入调用的函数

if(newValue>2004){this.\_year=newValue;this.edition+=newValue-2004;}}

}

});

//读取数据属性\_year

Var des=Object.getOwnPropertyDescriptor(book,”\_year”);

Alert(des.value); //2004

Alert(des.configurable); //false

//读取访问器属性year

Var des=Object.getOwnPropertyDescriptor(book,”year”);

Alert(des.value); //undefined 访问器属性没有这个描述符

Alert(des.configurable); //false