JavaScript

JavaScript的出现主要是用于处理网页中的前端验证

Node.js是用来写服务器的

ECMAScript 几家公司制定的JS标准

浏览器 JavaScript实现方式

FireFox SpiderMonky

Chrome v8引擎

JavaScript = ESMAScript（标准）+DOM（对象）+BOM（DOM标准）

**JS特点**

解释型语言

动态语言

基于原型的面向对象

<script type=”text/javascript”>

Alert(‘’);//控制浏览器弹出警告框

Document.write(“”);//让计算机在页面（body）输出一个内容

Console.log(“”);//向控制台输出一个内容

</script>

**JS代码编写的位置**

1. 可以将js代码写到js文件中，然后通过script标签引入（好处：代码复用）
2. 用script标签引入外部文件，标签里面不能写代码，写了也不允许
3. 如果本界面需要js代码，可以再写一个script标签用于写内部代码
4. Js代码可以写到标签的onclick属性中，点击按钮js代码就会执行（不推荐，因为结构和行为耦合，不方便维护）
5. Js代码可以写到超链接href属性中，点击超链接的时候会执行js代码

**基本语法**

多行注释

/\*

JS多行注释

\*/

单行注释

//JS单行注释

注意：

1. JS严格区分大小写
2. 每一条语句以;结尾（如果不写，浏览器会自动加，但是会耗费一些系统资源，有的时候系统还会加错，所以开发必须写）
3. JS会忽略多个空格和换行

**字面量和变量**

字面量：一些不可改变的值，比如：1 2 3 4 5 ，字面量可以直接使用

变量：

1. 变量可以保存字面量，变量的值任意可变
2. 在开发中，通常通过变量保存字面量
3. 可以通过变量对字面量进行描述

声明变量： var a

为变量赋值：a = 123

声明变量和变量赋值可以同时进行 var c

**标识符**

1. 含义：所有可以由我们自主命名的，例如变量名、函数名、属性名
2. 遵守规则
   1. 标识符可以含有字母、数字、\_、$
   2. 标识符不能以数字开头
   3. 标识符不能是ES中的关键字和保留字
   4. 标识符一般采用驼峰命名，例如helloWorld
   5. Utf-8含有的内容都可以作为标识符

**数据类型**

1. 含义：字面量的类型
2. 类型：
   1. 基本数据类型
      1. String 字符串
      2. Number 数值
      3. Boolean 布尔值
      4. Null 空值
      5. Undefined 未定义
   2. 引用数据类型
      1. Object对象

**字符串**

1. 需要使用引号
2. 单引号双引号都可以，但不能混着用 var a =”12’（不可）
3. 引号不能嵌套var a = ‘tom say:’hello’ ’(不可)单引号里面不能放单引号，双引号里面不能放双引号
4. 如果非要嵌套可以使用\作为转义字符
   1. \”表示 ”
   2. \’表示’
   3. \n表示换行
   4. \t表示tab
   5. [\\表示\](\\\\表示\\)
5. 输出的是变量还是字符串要区分
   1. Alert(‘str’)字符串str
   2. Alert(str)变量str

**Number**

1. JS中所有的数值都是Number类型，包括整数、浮点数（小数）
2. 数字中的最大值 Number.MAX\_VALUE 1.7976931348623157e+308
3. 数字中的最小正值 Number.MIN\_VALUE 5e-324
4. 使用Number表示的数字超过了最大值，则会返回一个
   1. Infinity 表示正无穷
   2. -Infinity 表示负无穷
   3. 使用typof检查Infinity也会返回number
      1. a = Infinity； console.log(typeof a)
5. NaN是一个特殊的数字，表示Not A Number，用typeof检查一个NaN也会返回number
6. 关于数字的计算：
   1. 在JS中整数的运算基本可以保证精确
   2. 使用JS进行浮点运算（0.1+0.2），可能会得到一个不精确的结果

**Boolean布尔值**

1、只有两个值true、false

2、用typeof检查一个布尔值，会返回Boolean

3、主要用作逻辑判断

**Null和Undefined**

1. Null类型的值只有null
   1. Null值表示一个为空的对象
   2. 用typeof检查null，会返回object
2. Undefined类型的值只有undefind
   1. 声明完变量，但不给变量赋值，变量的值就是undefined
   2. 用typedof检查一个undefined时也会返回undefined

**强制类型转换**

1. 含义：指将一个数据类型强制转换为其他的数据类型
2. 分类
   1. 转成String
   2. 转成Number
   3. 转成Boolean

**其他数据转成String**

1. 调用toString方法 a.toString();
   1. 转换结果：原值加引号
   2. Null和undefined这两个值没有toString()方法，如果调用会出错
   3. Number调用toString()，可以指定进制
      1. a = 255 a = toString(2) => a=11111111
2. 调用String()函数 String(a)
   1. 对于number和boolean来说和调用toString()没什么不同
   2. 对于null和undefined
      1. Null转换为”null”
      2. undefined转换为”undefined”

**其他数据转成Number**

1. 使用Number()函数
   1. 字符串
      1. 纯数字的字符串：直接转 “123”=>123
      2. 有非数字的字符串：则转换为NaN（不是一个数字）
      3. 空串或者全空格的串：转成0
   2. 布尔值
      1. True 1
      2. False 0
   3. Null（空对象） 0
   4. Undefined（定义了未赋值的变量） NaN
2. 专门使用于将字符串转换为数字（100px这种情况下使用）
   1. parseInt()把字符串转换为一个整数
   2. ParseFloat()把字符串转换为浮点数

**其他进制的数字**

1、16进制的数字，以0x开头

2、8进制的数字，以0开头 070（有些浏览器当成8进制，有些浏览器当成10进制）

解决方法： a = parseInt(a,10)

3、2进制的数字，以0b开头

注意：并不是所有的浏览器都支持

**其他数据转换成Boolean**

1. 使用Boolean()函数
   1. 数字
      1. 除了0和NaN，其余都是true
         1. 负数，例如-12也是true
   2. 字符串
      1. 除了空串，其余的都是true
         1. 注意是空串””，如果里面有空格，例如” ”就是true

**运算符**

1. 含义：通过运算符可以对一个或多个值进行运算
   1. 例如：typeof 获得一个值的类型 返回方式：字符串 内容：number、string、boolean、undefined、object

**算术运算符**

1. 对非Number类型的值进行运算：转成Number再运算
2. 任何值和NaN做运算都得NaN
3. 加运算+
   1. 对两个number进行加法运算：返回结果（优先级低）
   2. 对两个字符串进行加法运算：拼串（优先级高）12+”1”=121
   3. 任何值和字符串做加法：先转换成字符串，再拼串
4. 减运算-
5. 乘运算\*
6. 除运算/
7. 取余运算%

注意：

转换String新方法： 加空串””

原理：任何类型的值和字符串相加都可以转换为String

var c = 12; c=c+”” => c=”12”

转换Number新方法：-0或\*1或/1

原理：任何类型的值做-\*/运算的时候都会自动转换成Number

**一元运算符**

+正号 对数字不会产生任何影响

-负号 对数字的符号取反

对于非Number的值

1. 先转换成Number，再运算
2. 转换Number新方法：对其他的数据类型使用+

**自增和自减**

自增 ++

1. 效果：使变量+1
2. 分类
   1. a++ ++a
      1. 同：使原值+1
      2. 异：a++ 的值为自增前的值 ++a的值等于自增后的值

自减 --

1. 效果：使变量-1
2. 分类
   1. a-- --a
      1. 同：使原值-1
      2. 异：a-- 的值为自减前的值 --a的值等于自减后的值

**逻辑运算符**

！非

1. ！可以做非运算：即对一个布尔值取反，即true变false，false变true
2. 对一个值两次取反，则其不会变化
3. 对非布尔值取反：先转布尔值，再取反
4. 转换布尔值方法：！！a（取两次反）

&&与

1. 与运算
2. 运算规则
   1. 只要有一个false就返回false
   2. 第一个值为false，则不会扫第二个值

||或

1. 或运算
2. 运算规则
   1. 只要有一个true，返回true
   2. 第一个为true，则不会检查第二个值

&&和||非布尔值的情况

总体：先转布尔值，再运算

&&（进行到哪儿，返回对应的值）

1. 第一个值为true，返回第二个值
2. 第一个值为false，则直接返回第一个值

||（进行到哪儿，返回对应的值）

1. 第一个值为true，直接返回第一个值
2. 第一个值为false。返回第二个值

**赋值运算符**

= 右侧值赋值给左侧变量

+= a+=5 a = a + 5

-= a-=5 a = a - 5

\*= a\*=5 a = a \* 5

/= a/=5 a = a / 5

%= a%=5 a = a % 5

**关系运算符**

1. 作用：关系成立返回true，关系不成立返回false

包含：>、>=、<、<=

非数值的情况

1. 转换成数字后比较
2. 字符串：分别比较字符串的unicode编码
   1. 一个情况：”11”<”5”进行数据比较，一定要转换成数字
3. 任何值和NaN作比较都是false

**编码**

在js中的写法：\u四位编码(16进制)

在网页中的写法：&#四位编码对应的10进制编码;

**相等运算符**

== 比较两个值是否相等，相等返回true，不相等返回false

比较类型不同，会自动转换成相同类型

！= 比较两个值是否不相等，不相等返回true，相等返回false

比较类型不同，会自动转换成相同类型

=== 比较两个值是否全等

不会做类型转换

！==比较两个值是否不全等

不会做类型转换

做类型转换时的特例

Null == 0 false（扫描是从左到右扫的，0无法转换成null）

Undefined == null true 因为undefined衍生自null

NaN == NaN NaN不和任何值相等，包括他自己（不是一个数值的两个值可不一定一样，比如”hello”,”你好”）

判断一个值是否是NaN，可以调用isNaN()

**条件运算符（三元运算符）**

1. 语法：条件表达式？语句1：语句2
2. 执行流程
   1. 对条件表达式求值 为true，返回语句1。为false，返回语句2
   2. 如果条件表达式求值 转换成布尔值后再运算

**运算符的优先级**

，运算符 在声明多个变量时使用 var a=1，b=2，c=3

不清楚的优先级，可以使用（）来改变优先级

**代码块**

使用{}来为语句进行分组

每组语句要么都执行，要么都不执行

代码块只有分组的作用

**流程控制语句**

1. 含义：可以控制程序执行的流程
2. 分类
   1. 条件判断语句
      1. 条件成立执行语句，条件不成立语句不执行
      2. If语句 if（条件表达式）{语句...}
      3. if只能控制紧随其后的一条语句
      4. 控制多条语句：把语句放在一个代码块中{}
   2. 条件分支语句
   3. 循环语句

**If语句**

语法一：

If（条件表达式）{

语句...

}

语法二：

If（条件表达式）{

语句...

}else{

语句...

}

特点：条件表达式为true，执行if后的语句。条件表达式为false，执行else后的语句

语法三：

If（条件表达式）{

语句...

}else if（条件表达式）{

语句...

}else if（条件表达式）{

语句...

}else{

语句...

}

特点：

1. 从上到下依次对条件表达式进行判断，为true，执行当前语句，为false，向下判断，都不满足，执行最后一个else语句
2. 一旦有一个代码块被执行，结束语句

**条件分支语句（switch语句）**

语法：

switch（条件表达式）{

case 表达式：

语句...

break；

case 表达式：

语句...

break；

case 表达式：

语句...

break；

default：

语句...

break;

}

执行流程：

1. 依次将case后的值和条件表达式的值进行全等比较，为true，从当前case开始执行，直到遇到break停止。如果为false，继续向下比较，如果所有的case比较完了，还是false，则只执行default后的语句
2. Switch语句和if语句可以互换使用

**循环语句**

1. 含义：通过循环语句可以反复执行一段代码多次
2. 分类：
   1. While循环
   2. Do...while循环
   3. For循环

**While循环**

语法：

While（条件表达式）{

语句...

}

特点：

1. 对条件表达式判断，如果值为true，则执行循环体
2. 循环体执行完后，继续对表达式进行判断，如果为true，继续执行循环体，如果为false，则终止循环

**Do...while循环**

语法：

Do{

语句...

}while(条件表达式)

特点：

1. 先执行循环体，后对条件表达式进行判断，为true，继续执行循环体，为false，则终止循环

While和do.. while比较

同：功能类似，都是实现对语句的反复调用

异：do..while能保证循环体至少执行一次，而while不能

死循环：在循环中，条件表达式的值写死为true

创建循环的步骤：

1. 初始化一个变量。i=0
2. 在循环中设置一个条件表达式。i<10
3. 定义一个更新表达式，每次更新初始化变量。i++

练习中的新函数

Prompt(“请输入小明的期末成绩”)

可以弹出一个提示框，提示框里面有个文本框，返回的是用户输入的内容

**For语句**

特点：提供专门的位置放三个表达式

1. 初始化表达式
2. 条件表达式
3. 更新表达式

语法：

For（初始化表达式；条件表达式；更新表达式）{

语句...

}

For循环执行的流程：

1. 执行初始化表达式，初始化变量
2. 执行条件表达式，判断是否能进入循环
   1. 为true，执行循环体
   2. 为false，终止循环
3. 执行更新表达式，后执行条件表达式，判断是否能进入循环，即循环2

特点：

1. for循环中的三个部分都可以省略，写在外部
2. For（；；）此时循环是个死循环，会一直执行下去

**嵌套的for循环**

例子：通过for循环输出图形

\*

\*\*

\*\*\*

for(var i=0;i<5;i++){控制图形的高度：列

For(var j=0;j<i;j++){控制图形的宽度：行

Document.write(“\*&nbsp;”)

}

}

**Break和continue**

Break和continue特点

1. 可以用来退出switch或循环语句
2. 不能在if语句中使用
3. 会停止离他最近的那个循环语句
4. 可以为循环语句创建一个label，标识当前的循环 label：循环语句
5. 使用break的时候，后面加个label，break就会结束指定的魂环 break label；
6. Continue是用来跳过当次循环，continue只对离他最近的循环起作用

练习中的新函数

Console.time(“test”)开启一个名为test的定时器

Concole.timeEnd(“test”)终止一个名为test的定时器

会在控制台输出机器执行代码的时间

Math.sqrt()可以对一个数开方

**对象（Object）**

优点：

基本数据类型（String、Number、Boolean、Null、Undefined）：值和值之间没有任何的联系

对象：是一种复合的数据类型，在对象中可以保存多个不同数据类型的属性

对象分类：

1. 内建对象：ES标准中定义的对象，任何ES中都可用
   1. 如：Math、String、Number、Boolean、Function、Object
2. 宿主对象：由JS的运行环境提供的对象，也就是浏览器提供的对象
   1. 如BOM、DOM
3. 自定义对象：由开发人员自己创建的对象

创建对象：

1. 语法：var obj = new Object（）；
2. 使用new关键字调用的函数，是构造函数constructor
3. 构造函数是专门用来创建对象的函数

向对象中添加属性：

1. 语法：obj.name=”sam” 对象.属性名=属性值

读取对象中的属性

1. 语法：obj.name 对象.属性名
2. 如果读取对象中没有的属性，不会报错而是会返回undefined

修改对象的属性值

1. 语法：obj.name=”tom” 对象.属性名 = 属性值

删除对象的属性

1. 语法：delete obj.name delete 对象.属性名

**属性名和属性值**

属性名：

1. 对象中的属性名不要求遵守标识符的规则
2. 如果使用特殊的属性名，需要用另一种方式，读取的时候也需要采取这种方式
   1. 语法：对象[“属性名”]=属性值 obj[“789”]=123、console.log(obj[“789”])

属性值：

1. 属性值可以是任意的数据类型，甚至可以是一个对象

In 运算符：通过该运算符可以检查一个对象中是否含有指定的属性，有则返回true，没有则返回false

语法：”属性名” in 对象 console.log(“name” in obj)

**基本和引用数据类型**

基本数据类型：String、Number、Boolean、Null、Undefined

引用数据类型：Object

内存分类：

栈内存

1、JS中的变量都是保存到栈内存中的

2、基本数据类型的值直接在栈内存中存储

3、栈内存中的值和值之间是独立存在的，修改一个变量不会影响其他的变量

堆内存

1、对象是保存到堆内存中的

2、每创建一个对象，都会在堆内存中开辟一个新的空间

3、对象变量中保存的是该对象的内存地址，如果两个变量保存的是同一对象的引用，当通过一个变量修改属性时，另一个也会受到影响。

基本和引用数据类型特点比较

1、当比较两个基本数据类型，比较的是两个基本数据类型的值

2、当比较两个引用数据类型，比较的是对象的内存地址，如果两个对象里的内容是一模一样的，但是地址不同，也会返回false

**对象的字面量**

1、使用对象的字面量创建对象 var obj = {}；

2、在创建对象的时候，可以直接指定对象的属性

3、语法{属性名：属性值，属性名：属性值...}

Var people = {

Name：tom，

Age：18

Sex：”女”

}

4、属性名可以加引号也可以不加，如果使用特殊的名字则必须加引号

5、最后一个属性不要写。

**函数（function）**

特点：

1. 函数也是一个对象
2. 函数里面可以封装一些功能，并且在需要的时候执行这些功能
3. 用typeof检查一个函数对象的时候，会返回function

创建一个函数对象(用构造函数)

Var fun = new function（“ 语句”）;//不会立即执行

调用函数

Fun（）;

创建一个函数（用函数声明）

Function fun2(){

语句...

}

创建一个函数（用函数表达式）

Var fun3 = function(){

语句...

}

**函数的参数**

函数的（）中可以指定一个或多个形参

声明形参相当于在函数内部声明了对应的变量

但是并没有赋值

Function sum（a,b）{//相当于var a var b//形参

Return a+b

}

在调用函数时，可以在()中指定实参

实参会赋值给函数中对应的形参

Sum（2，3）

注意：

1. 调用函数的时候不会检查实参的类型，函数的实参可以是任意的数据类型
2. 调用函数的时候也不会检查实参的数量
   1. 多余的实参不会被赋值
   2. 实参的数量少于形参，没有对应的形参将是undefined
3. 传的参数多的话可以打包成一个对象，通过对象传递
4. Mianji（）调用函数
5. Mianji函数对象

**返回值**

可以使用return来设置函数的返回值

Return后面的值将会作为函数的执行结果返回

在函数中return后面的语句都不会执行

Return后面不跟任何值/不写return 也会返回undefined

Return后可以跟任意类型的值

Return、break和continue

Return可以结束整个函数

Continue跳过当次循环

Break可以退出当次循环

Return的返回值也可以是一个函数，从而实现函数的嵌套

Function fun3（）{

Function fun4(){

}

Return fun4;

}

函数的调用fun3()();

**立即执行函数**

含义：函数定义完，立即被调用的函数

立即执行函数往往只执行一次

(function(){

语句...

})();

没有函数名，用括号包起来，需要立即用

**对象**

创建一个对象

Var obj = new Object（）;

在对象中添加属性

Obj.name = “孙悟空”;

对象的属性可以是函数

Obj.sayName = function(){

}

调对象里面的方法

Obj.sayName();

我们称在window身上的函数叫函数，称在自己写的对象上面的函数叫方法（只是名称的区别）

**枚举对象中的属性**

语法

For(var 变量（n） in 对象){

}

对象中有几个属性，循环体就会执行几次

变量里面存的是属性名（n）

Obj[n]可以取到属性值

**作用域**

作用域指一个变量的作用的范围

1. 全局作用域
   1. 直接编写在script标签中的JS代码
   2. 页面打开时创建，页面关闭时销毁
   3. 全局对象window，代表的是一个浏览器的窗口
   4. 创建的变量都会作为window对象的属性保存
   5. 创建的函数都会作为window对象的方法保存
   6. 全局变量在页面的任意部位都可以访问的到
2. 函数作用域
   1. 调用函数时创建，函数调用完毕时销毁
   2. 在函数内声明的值

**变量的提前声明**

使用var关键字声明的变量，会在所有的代码执行之前被声明

不适用var关键字，变量不会被声明提前

**函数的声明提前**

语法：function 函数名(){}

他会在所有代码执行之前被创建

使用函数表达式创建的函数，不会被声明提前

**函数作用域**

1. 调用函数时创建函数作用域，函数执行完毕后，函数作用域被销毁
2. 每调用一次函数都会创建一个新的函数作用域
3. 在函数作用域中可以访问到全局作用域的变量，反之不行
4. 在函数作用域操作一个变量时，首先会在自身的作用域中寻找，有则直接用，没有则向上一级的作用域中寻找，直到找到全局作用域，如果在全局作用域中还没找到，则会报错ReferenceError
5. 如果在函数中想要越过自身作用域访问全局变量，可以使用window对象
6. 在函数作用域中也有声明提前的特性
7. 在函数中，不适用var声明的变量都会成为全局变量
8. 定义的形参相当于在函数作用域中声明了变量

**Debug**

在网页中可以调试数据

F12

Sources

Debug.html

打断点

分步执行

Add selected text to watches//选择好要监视的属性

**This**

解析器在调用函数时每次都会向函数内部传递一个隐含参数this，使得this指向一个对象

根据函数的调用方式的不同，this会指向不同的对象

1. 以函数的形式调用时，this会指向window
2. 以方法的形式调用时，this指向调用方法的对象

通俗说：this被谁调用就指向谁

**使用工厂的方式来创建对象**

**使用函数声明的方式**

缺点：使用的构造函数都是Object，无法区分多种不同类型的对象

**使用构造函数的方式（推荐）**

构造函数特点

1. 构造函数就是一个普通的函数，创建构造函数习惯首字母大写
2. 构造函数调用方式不同，需要使用new关键字来调用
3. 执行过程
   1. 立刻创建一个新的对象
   2. 将新建的对象设置为函数中的this，在构造函数中可以使用this来引用新建的对象
   3. 逐行执行函数中的代码
   4. 将新建的对象作为返回值返回
4. 使用构造函数创建的对象，我们称为一类对象

This的情况：

1. 以函数的形式调用，this是window
2. 以方法的形式调用，谁调用方法this就是谁
3. 以构造函数的形式调用时，this就是新创建的那个对象

//创建构造函数

Function Person（name）{

This.name = name;

This.sayName = function(){

Alert(this.name);

}

}

Function Dog(){

}

//创建构造函数对应的实例对象

Var per = new Person(“sam”)

Var dog = new Dog();

注意：

用instanceof可以检查一个对象是否是一个类的实例

对象 instanceof 构造函数

Per instanceof Person

如果是则返回true，不是则返回false

所有的对象都是Object的后代，任何对象和Object做instanceof检查时都会返回true

缺点：在构造函数里面直接写sayName的方法，会造成调用10000次，创建10000个相同的方法的情况，造成浪费

解决方法1

写一个函数sayName

Function fun(){

Alert(this.name)

}

Function Person(name){

This.name = name;

This.sayName = fun

}

缺点：函数是不能重名的，很不方便

解决方法2

往原型上面添sayName的方法

Person.prototype.sayName = function(){

}

**原型（prototype）**

1、当我们每创建一个函数（主要是构造函数），解析器都会向函数中添加一个属性prototype

2、使用构造函数所创建的对象都会有一个隐含的属性，指向构造函数的原型对象，即\_\_proto\_\_

Person.prototype == per.\_\_proto\_\_

3、原型对象是一个公共的区域，所有同一个类的实例都可以访问到原型对象

4、当我们访问对象的属性或者方法的时候，它会先在自身中找，有则用，没有则去原型对象中找，如果有则用，如果没有就去原型的原型中寻找，直到找到Object的原型对象，如果还是没有，则为返回undefined

5、注意：

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

“name” in per

可以使用对象的hasOwnProperty()来检查对象自身中是否含有该属性

这个函数只有对象自身含有属性时，才会返回true

Per.hasOwnProperty(“name”)

**ToString方法**

如果希望在输出对象的时候不输出[object object]，可以在构造函数的原型上添加一个toString方法，覆盖object的toString方法

Person.prototype.toString = function(){

}

Per.toString()

**垃圾回收**

垃圾指的是一个对象没有任何变量或者属性对它进行引用，这个对象我们无法操作，这个对象就是垃圾。

JS有自动的垃圾回收机制

我们需要做的就是把不再使用的对象设置为null

Var obj = new Object（）;

Obj = null

对象分类：内建对象、宿主对象、自定义对象

**数组（Array）**

特点：

1. 数组也是一个对象
2. 主要功能：用来存储一些值
3. 数组是用数字作为索引来操作元素的

索引：从0开始的整数

**创建数组对象**

Var arr = new Array（）;

使用typeof检查一个数组时，会返回object

**在数组中添加元素**

Arr[0]=10;

Arr[1]=33

**读取数组中的元素**

Arr[0]

获取数组的长度：可以使用length属性来获取数组的长度

**Length属性：**

对于连续的数组，使用length可以获取到数组长度

对于非连续的数组，使用length获取的是最大的索引+1

**修改length**

Arr.length = 2

如果修改的length大于原长度，多出来的部分会空出来

如果修改的length小于原长度，多出来的元素会被删掉

向数组的最后一个位置添加元素

Arr [arr.length] = 70;

使用字面量来创建数组

var arr = []

在创建数组时，可以指定数组中的元素

Var arr = [1,3,4]

使用构造函数创建数组时，也可以同时添加元素

Var arr = new Array(1,2,3)

注意：

1、Var arr = [1]创建的数组只有一个元素1

2、Var arr = new Array(1)创建长度为1的数组

3、数组中的元素可以是任意类型的数据（可以是对象，函数，数组）

4、数组的元素为数组称为二维数组

**数组的方法**

Push()

该方法可以向数组的**末尾**添加一个或多个元素，并**返回**数组的**新的长度**

Pop()

该方法可以删除数组的最后一个元素，返回值是被删除的元素

Unshift()

向数组开头添加一个或多个元素，并返回新的数组长度

在开头插入元素后，其他元素的索引会依次进行调整

Shift()

可以删除数组的第一个元素，返回值为被删除的元素

**数组的遍历**

For(var i=0 ; i<arr.length;i++){

Console.log(arr[i]);

}

**forEach()**

使用须知：

这个方法只支持IE8以上的浏览器，IE8及其以下的浏览器都不支持

需要一个函数作为参数，但是函数是我们创建的，是浏览器进行调用的

因此这种函数被称为回调函数

数组有几个元素，函数就会执行几次

浏览器会在回调函数中传递三个参数

第一个：当前正在遍历的元素

第二个，当前正在遍历的元素的索引

第三个，正在遍历的数组

Arr.forEach(function(value,index,obj){

})

**slice()和splice()方法**

Slice()

1. 可以用于从数组中提取指定元素
2. 该方法不会改变元素数组，而是将截取的元素封装到一个新数组中返回
3. Slice（2，3）
   1. 2为截取开始的位置的索引，包含2
   2. 3为结束的位置的索引，不包含3
   3. 第二个参数可以省略不写，例如slice（2），此时会截取2之后的所有元素
   4. 索引也可以传递负值，例如slice（2，-1），意思是截取从第二个开始到倒数第一个中间的值

Splice()

1. 可以用于删除数组中的指定元素
2. 使用splice会影响到原数组，会将指定元素从原数组中删除
3. 被删除的元素会作为返回值返回
4. 参数
   1. 第一个，开始位置的索引
   2. 第二个，表示删除的数量
   3. 第三个及以后，可以传递一些新的元素，这些元素会自动插入到开始位置索引前

**数组剩余的方法**

Concat（）

可以连接两个或多个数组，并将新的数组返回

该方法不会对原数组产生影响

Join（）

该方法可以将数组转换成一个字符串

该方法不会对原数组产生影响，而是将转换后的字符串作为结果返回

参数：参数就是连接字符串的连接符，如果不指定连接符，则默认使用，作为连接符

Reverse（）

该方法可以用来反转数组（前面的和后面的换位置）

该方法会直接修改原数组

Sort（）

可以用来对原数组的元素进行排序

也会影响原数组，默认会按照Unicode编码来进行排序

即使是对于纯数字的数组，使用sort（）排序时，也会按照Unicode编码来排序。所以排序11会小于2，得到错误的结果

所以需要我们自己指定排序的规则

Arr.sort(function(a,b)){

Return a-b//升序排列

Return b-a//降序排列

//a-b大于0，则元素会交换位置，小于等于0，元素不会交换位置

}

**函数的方法**

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.apply() fun.call()

共同点：

调用call（）和apply（）都会调用函数执行

第一个参数：对象 这个对象将会成为函数执行时的this

不同的：

Call（）可以传实参，第二个之后

例：fun.call（obj，2，3）

Apply（）可以传实参，可以封装到一个数组中统一传递

例：fun.apply（obj，[2,3]）

this的情况

1. 以函数形式调用，this永远是window
2. 以方法的形式调用，this是调用方法的对象
3. 以构造函数的形式调用，this是新创建的那个对象
4. 使用call和apply调用，this是指定的那个对象

**Arguments(调用函数时，浏览器传递的隐含参数)**

Arguments是类数组对象，可以用索引来操作数据

Arguments里面保存着实参

Arguments.length可以获取实参的长度

不定义形参，也可以用arguments来使用形参 arguments[0]

有一个属性callee，指向函数的对象 arguments.callee == fun

**Date对象**

创建Date对象

Var d = new Date()封装的是当前代码执行的时间

可以在创建对象的时候指定具体的时间

Var d2 = new Date（“2/18/2021 11:11:11”）

函数：

getDate（）获取当前对象是几日

getDay（）获取当前对象是周几

0周日，1周一

getMonth（）获取当前对象的月份

0一月1二月11十二月

getFullYear（）获取当前对象的年份

getTime（）获取当前对象的时间戳

时间戳：从格林威治标准时间的1970年1月1日，0时0分0秒到当前日期所花费的毫秒数（计算机底层保存的时间）

**Math对象**

Math对象不是一个构造函数，属于工具类，里面封装了数学运算相关的属性和方法

比如Math.PI表示圆周率

函数：

abs()可以用来计算一数的绝对值

Math.ceil()对一个数向上取整1.1 =>2

Math.floor()对一个数向下取整1.9=>1

Math.round()对一个数进行四舍五入

Math.random()可以用来生成一个0-1之间的随机数（不包括0和1）

生成0-x之间的随机数（包括0和x，并且是整数）

Math.round(Math.random()\*x)

生成x-y之间的随机数

Math.round(Math.random()\*(y-x)+x)

Math.max()获取多个数中的最大值

Math.min()获取多个数中的最小值

Math.pow(x,y)返回x的y次幂

Math.sqrt()对一个数进行开方

**包装类**

String()可以将基本数据类型字符串转换为String对象

Number()可以将基本数类型的数字转换成Number对象

Boolean()可以将基本数据类型的布尔值转换成Boolean对象

在实际应用中我们并不使用，但计算机会用

计算机的应用：

当我们对一些基本数据类型的值调用属性和方法，浏览器会临时使用包装类将基本数据类型转换成对象，再调用方法，之后立即转换成基本数据类型

**字符串的相关方法**

在底层字符串是以字符数组的形式保存的 [“H” ,”e”,”l”]

Str.length可以用来获取字符串的长度

Str.charAt()可以返回字符串中指定位置的字符，根据索引获取，例如Str.charAt(6)

Str.charCodeAt()用来获取指定位置字符的字符编码（Unicode编码），通过索引来获取

String.fromCharCode(0x2692)可以根据字符编码获取字符

Str.concat()可以用来连接两个或多个字符串，作用和+一样

Str.indexOf(“h”,1)可以检索一个字符串是否含有指定内容

有则会出现第一次检索到的索引，没有则返回-1

参数：

第一个：要检索的内容

第二个：开始查找的位置

Str.lastIndexOf()使用方法同indexOf，不同的是查找顺序是从后往前找

Str.slice()可以从字符串中截取出指定的内容，不会影响原字符串

参数：

第一个：开始位置的索引（包括）

第二个：结束位置的索引（不包括），第二个参数可以省略，省略之后会截取后面所有的数据

可以传递负数作为参数，负数是从后往前计算

Str.substring()可以用来截取一个字符串

基本同slice()

不同的是这个方法不能接受负值作为参数

如果传递了负值，默认使用0，其次这个函数还会自动调整参数的位置，保证第一个参数小于第二个参数

Str.substr()可以用来截取字符串，例如：str.substr(3,2)

参数：

1. 截取开始位置的索引
2. 截取的长度

Str.split()可以将一个字符串拆分成一个数组。例如：str.split(“,”)

如果传递空串作为参数，那么每个字符都会拆分成数组中的一个元素

Str.toUpperCase()将一个字符产转换成大写并返回

Str.toLowerCase()将一个字符串转换成小写并返回

**正则表达式**

1. 正则表达式用来定义一些字符串的规则
2. 正则表达式用途
   1. 检查一个字符串是否符合规则
   2. 将字符串中符合规则的内容提取出来
3. 创建正则表达式
   1. 语法： var 变量 = new RegExp（“正则表达式”,”匹配模式”）
      1. Var reg = new RegExp(“a”,”i”)
   2. 匹配模式
      1. i 忽略大小写
      2. g 全局匹配模式
4. 正则表达式的方法

Reg.test(“Ac”)

1. 使用字面量来创建正则表达式
   1. 语法：var 变量 = /正则表达式/匹配模式
      1. Var reg = /a/i
      2. Reg = /a|b/检查一个字符串中是否有a或b
      3. Reg = /[A-z]/检查一个字符串中是否有字母
      4. [ab] == a|b
      5. [a-z]任意小写字母
      6. [A-Z]任意大写字母
      7. [A-z]任意字母
      8. [0-9]任意数字
      9. Reg = /a[bde]c/检查字符串是否含有abc或adc或aec
      10. Reg=/[^ab]/除了ab之外都可以

**字符串和正则相关的方法**

Split()

将一个字符串拆分成一个数组

可以传递一个正则表达式作为参数

即使不指定全局匹配g，也会全部拆分

Str.split(/[A-z]/);

Search()

搜索字符串中是否含有指定内容，如果搜索到指定内容，则会返回第一次出现的索引，如果没有搜索到返回-1

可以接受一个正则表达式作为参数

Search只会查到第一个，即使设置全局匹配也没用

Str.search(/a[bef]c/)

Match()

可以根据正则表达式，从一个字符串中将符合要求的内容提取出来

默认只找第一个符合要求的内容

可以设置全局匹配，这样会匹配所有符合要求的内容

匹配到的内容会封装到一个数组中返回

Replace()

可以将字符串中指定内容替换成新的内容

参数：

第一个，被替换的内容，可以接受正则表达式

第二个，新的内容

默认换第一个，可以设置全局匹配

**正则表达式**

量词

通过量词可以设置一个内容出现的次数

例如：/a{3}/ 表示为aaa

两次只对它前面出现的一个内容起作用

例如：/(ab){3}/ 表示为ababab

/ab{1,3}c/ 表示出现1-3次

/ab{3,}/ 表示出现3次以上

+ 至少一个，相当于{1,}

\*

(未完但不做了)