JS中所有数字均用浮点数值表示。

JS中的运算符包括：+、-、\*、/、%，还有Math对象的属性，例如：Math.pow()、Math.round()等等。

下溢：运算结果无限接近零并比JS能表示的最小值还小的时候发生的一种情况。这种情况下，JS返回0.当一个负数发生下溢时，JS将返回一个特殊的值“负零”，这个值几乎和正常的0完全一样。JS程序员很少用到负零。

被零整除在JS中不报错，只是简单的返回无穷大或负无穷大。0/0没有意义，结果为NaN,无穷大除无穷大、给任意负数作开方运算或者算术运算符与并不是数字或无法转换为数字的操作数一起使用时都返回NaN。

由双引号定界的字符串也可以包含单引号。

JS中的字符串直接量：由单引号或双引号引起来的字符序列。反斜线和行结束符都不算是字符串直接量的内容。

如果想要在字符串直接量中另起一行，可以用转义字符\n。

转义字符：\b退格符，\n换行符，\\反斜线。

JS定义了RegExp()构造函数，用来创建表示文本模式匹配的对象。这些模式被成为“正则表达式”。

JS中采用Perl中的正则表达式语法。String和RegExp对象均定义了正则表达式进行模式匹配和查找与替换的函数。

RegExp和Date一样，是具有实用API的特殊对象，依然具有直接量写法。

在两条斜线之间的文本构成了一个正则表达式直接量。第二条斜线之后也可以跟随一个或多个字母，用来修饰匹配模式的含义。

例：/^HTML/：匹配以HTML开始的字符串。

被转换为false:undefined、null、0、-0、NaN、空字符串。null==undefined// null !==undefined

JS解释器启动时（web浏览器重新加载新页面的时候），它将创建一个新的全局对象，并给它一组定义的初始属性：全局属性、全局函数、构造函数、全局对象。

全局对象的初始属性并不是保留字，但它们应当当作保留字来对待。

存取字符串、数字、或布尔值的属性时创建的临时对象成为包装对象。包装对象只是偶尔用来区分字符串值和字符串对象、数字和数值对象以及布尔值和布尔对象。

可以通过String()、Number()、Boolean（）构造函数来显示创建包装对象。

JS中的原始值不可以改变，对象的值可以改变。原始值的比较的是值，对象比较的是引用。

toString()方法null和undefined不能使用，该方法和String()方法返回结果一致。

“！”运算符：将操作数转换为布尔值并取反。

Number三种方法:toFixed():根据小数点后的指定位数将数字转换为字符串，从不使用指数记数法。

toExponential():使用指数记数法将数字转换为指数形式的字符串，小数点前只有一位，其中小数点后的位数由参数决定。

toPrecision():根据指定的有效数字位数将数字转换为字符串。如果有效数字的位数小于整数部分的位数，则转换为指数形式。

parseInt()函数：只能解析整数，parseFloat():可以解析整数和浮点数。

parseInt()函数：可以接收第二个可选参数，这个参数指定数字转换的基数，合法的取值范围是2-36。例：parseInt("11",2);// =>3(1\*2+1)不懂。

new Boolean(false)是一个对象而不是原始值，它将转换为true。

valueOf()方法返回的是一个对象。

“+”运算符：1>可以进行数字加法和字符串连接操作。2>如果其中一个操作数为对象，则JS将使用特殊的方法将对象转换为原始值。

“==”：如果将对象和一个原始值进行比较，则转换将会遵照对象到原始值的转换方式进行。

如果重复声明带有初始化器，那么这和一个简单的赋值语句没有什么两样。JS无法读取一个没有声明的变量的值。

在函数体内，局部变量的优先级高于同名的全局变量。

如果在函数内声明的一个局部变量或者函数参数中带有的变量和全局变量重名，那么全局变量会被遮盖。

例：var scope="global";function checkscope(){var scope="local;return scope;} checkscope();//local

函数定义可以嵌套的，每个函数有自己的作用域，因此会出现几个局部作用域嵌套的情况。

如果没有使用严格模式并给一个未声明的变量赋值,JS会自动创建一个全局变量，此时全局对象的正常的可配置属性。

JS全局变量是全局对象的属性。this关键字引用全局对象，不能引用局部变量中存放的对象。

定义一个函数时，保存了一个作用域链。当调用该函数时，创建一个新的对象来存储它的局部变量，并将这个对象添加至保存的那个作用域链上，同时创建一个新的更长的表示函数调用作用域链。

嵌套函数，每次调用外部函数时，内部函数会重新定义一遍。

静态类型检查语言会在编译期间进行检查，动态类型检查语言会在执行期间进行检查。

一种语言可以同时做静态类型检查和动态类型检查，如果检查失败了，程序会抛出错误或者异常。

undefined出现的场景：未初始化的变量、缺失的参数、访问不存在的属性、函数没有显示的返回值，函数会隐式返回undefined。

null出现的场景：null是原型链最顶端的元素、字符串没有匹配到正则表达式结果。

检测null和undefined通过严格相等“===”检查。

包装：对原始值增加属性。去包装：使用值之前需要去包装。

包装：通过调用包装构造函数例：new Number(123);new String("abc");

去包装：通过valueOf()。例： new String("abc").valueOf();//abc

注：将包装对象转换为原始值时只能正确地提取出数字和字符串，而布尔值不能。

ToPrimitive():将值转换为原始值。函数原型：ToPrimitive(input，PreferredType),PreferredType表明转换后的类型。

valueOf的默认实现会返回this(调用者)，toString()的默认返回类型信息。

var empty={};empty.valueOf();//empty empty.toString();//"[Object,Object]"