原始表达式（不再包含其他表达式）是表达式的最小单位。JS的原始表达式包含：常量、直接量、关键字、变量。

直接量是直接在程序中出现的常数值。JS的一些保留字构成了原始表达式例如：true、false等。

数组初始化表达式是可以嵌套的。例：var matrix=[[1,2],3,4].数组初始化表达式也称为：“数组直接量”。对象初始化表达式也称为“对象直接量”。

JS对数组初始化表达式进行求值的时候，数组初始化表达式中的元素表达式会各自计算一次。

函数定义表达式也可以称为：函数直接量。

函数调用表达式：是一种调用（或执行）函数或方法的语句。

对象创建表达式：创建一个对象并调用一个函数（这个函数称作构造函数）初始化新对象的属性。对象创建表达式和函数调用表达式非常类似，只是对象创建表达式之前多了一个关键字new。

例：new Object(); new Point(2,4);如果一个对象创建表达式不需要传入任何参数给构造函数，那么圆括号可以省略。new Object;

运算符可以用关键字表示，例：delete,instanceof/。delete运算符删除一个属性就像给这个属性赋值为undefined。

运算符优先级高先执行。运算符的优先级可以通过显示使用圆括号重写。一元运算符、赋值和三元条件运算符都具有从右至左的结合性。

算术运算符的操作数如果是NaN,算术运算结果也是NaN。0/0结果是NaN

“+”运算符：操作数如果是对象则先转换为原始类型值。日期对象通过toString()方法进行转换，其他对象通过valueOf()。然而多数对象不具备valueOf()方法，因此会通过toString()方法转换。

++x；不永远和x=x+1;等价例如：x='1'x为字符串时，++从不进行字符串拼接操作。

位运算符：^按位异或，异或指的是第一个操作数为true或第二个操作数为true，但不能同时为true。&按位与、|按位或、~按位非、

<<左移：移动的位数由第二个操作数指定。新的第一位用0补充。a<<1 a的第一位变成第二位。

>>带符号右移：填补在左边的位由原操作数的符号决定。正数最高位补0，负数最高位补1。

将一个操作数右移一位相当于用它除以2（忽略余数）。

>>>(无符号右移)：最高位总是补0。

in运算符：该运算符希望它的左操作数是一个字符串或可以转换为字符串，希望它的右操作数是一个对象。如果右侧的对象拥有一个名为左操作数值的属性名，那么返回true。

var point={x:1,y:2}; "x"in point;//true。

instanceof运算符：希望左操作数是一个对象，右操作数标识对象的类。如果左侧的对象是右侧类的实例，则表达式返回true。

例：var d=new Date(); d instanceof Date;//true理由：d是由Date()创建的。

如果instanceof的左操作数不是对象，返回false。右操作数不是函数，则会抛出一个类型错误异常。

eval()是一个全局函数，但是被当成运算符来对待。

eval()存在的问题是：用于动态执行的代码通常是不能分析。故，一个函数调用了eval（）,那么解释器将无法对这个函数做进一步优化。eval()可以被赋予其他的名字。

例：var f=eval; var g=f;故解释器将无法放心的优化任何调用g()函数。

eval()只有一个参数，如果传入的参数不是字符串，直接返回该参数。如果是字符串，则把字符串当作JS代码进行编译，如果编译失败则抛出一个语法错误异常。

如果编译成功，则开始执行这段代码，并返回字符串中的最后一个表达式或语句的值。

如果最后一个表达式没有值，则返回undefined。如果字符串抛出一个异常，这个异常将把该调用传递给eval（）。

eval()使用了它的变量作用域环境，换句话说，它查找变量的值和定义新变量和函数操作和局部作用域代码完全一样。

如果在最顶层代码中调用了eval()，它会作用于全局函数和全局变量。

ECS5严格模式下，调用eval()时，或eval()执行的代码以“use strict”指令开始，这里的eval()是私有上下文环境中的局部eval。

在严格模式下，eval执行的代码段可以查询或者更改局部变量，但不能在局部作用域中重新定义新的变量或函数。

严格模式下，eval列为保留字，这样eval()更像一个运算符。不能用一个别名覆盖eval()函数。并且变量名、函数名、函数参数或者异常捕获的参数都不能取名为eval。

void:一元运算符，出现在操作数之前，操作数可以是任意类型。操作数会照常计算，但忽略计算结果并返回undefined。void运算符经常在客户端的URL中。

例：<a href="javascript:void window.open();"></a>通过给<a>的onclick绑定一个事件处理程序要比在href中写“javascript:URL”要更加清晰。当然，这样的化，void操作符可有可无。

“，”运算符操作数类型可以是任意的，首先计算左操作数，然后计算右操作数，最后返回右操作数值。

复合语句：只需用花括号将多条语句括起来就是复合语句。

语句块注意事项：语句块的结尾不需要分号。

var和function都是声明语句，它们声明或定义变量或函数。var声明的变量是无法通过delete删除的。

语句：if语句、else if语句、switch语句、while循环、do while循环、for循环、for in循环。在每一个case语句块的结尾处都使用了关键字break。

for(variable in object) statement。variable通常是一个变量名，也可以是一个可以产生左值的表达式或者一个通过var语句声明的变量。总之必须是一个适用于赋值表达式左侧的值

object是一个表达式，这个表达式的计算结果是一个对象。同样，statement是一个语句或语句块，它构成了循环的主体。

for/in循环是用来更方便地遍历对象属性成员。在执行该循环过程中，JS解释器首先计算object表达式。如果表达式为null或undefined，JS解释器将会跳过循环并执行后续代码。

如果表达式等于一个原始值，这个原始值将会转换为与之对应地包装对象。JS会依次枚举对象的属性来执行循环。然而每次循环之前，JS会先计算variable表达式的值，并将属性名赋值给它。

每次循环都会计算variable的值。

在下列情况下，枚举的顺序取决于具体的实现（并且是非交互的）：对象继承了可枚举属性；对象具有整数数组索引的属性；使用delete删除了对象已有属性；

使用Object.defineProperty或者类似的方法改变了对象的属性。

JS中的跳转语句：break语句、continue语句、return语句让解释器跳出函数体的执行并提供本次调用的返回值。

throw语句触发或者‘抛出’一个异常，与try/catch/finally语句一同使用，这些语句指定了处理异常的代码逻辑。

当抛出一个异常时，程序将跳转至最近的闭合异常处理程序，这个异常处理程序可以在同一个函数或者更高层的调用栈中。

一个Error对象有一个name属性表示错误类型，一个message属性用来存放传递给构造函数的字符串。

如果不存在处理异常的局部catch从句，解释器会首先执行finally中的逻辑，然后向上传播这个异常，直到找到能处理这个异常的catch从句。

with语句用于临时扩展作用域链，严格模式下禁止使用with语句，非严格模式下不推荐使用。with语句已经被废弃。

debugger语句用来产生一个断点，JS代码执行时会停止在断点的位置，可以使用调试器输出变量的值、检查调用栈等。

debugger语句不会启动调试器。但如果调试器已经在运行中，这条语句才会产生一个断点。

‘use strict’是ECS5引入的一条指令，并不是语句。

指令并不包含任何语言的关键字，仅仅是一个包含一个特殊字符串值直接量的表达式。

undefined和null宽松相等。

for-in遍历数组得到的结果是索引，而不是数组元素。