1. **词法结构**

**2.8.2显示类型转换**

做显示类型转换最简单的方法就是使用Booklean().Number().String(),Object()函数，当不通过new运算符调用这些函数时，它们会作为类型转换函数并按照表3-2描述的规则做类型转换。

|  |  |
| --- | --- |
| Number(“3”) | =>3 |
| String(flase) | =>”false”或使用false.toString() |
| Boolean([]) | =>true |
| Object(3) | =>new Number(3) |

需要注意的是，除了null或undefined之外的任何数值都具有tostring（）方法，这个方法的执行结果通常和String（）方法的返回结果一致

toFixed() toExponential() toPrecision()这三个都四舍五入或填充0

|  |  |
| --- | --- |
| Var n=123456.789 |  |
| n.tofixed(0); | =>123457 |
| n.tofixed(2); | =>123456.79 |
| n.tofixed(5); | =>123456.78900 |
| n.toExponential(1); | =>1.2e+5 |
| n.ToExponential(3); | =>1.235e+5 |
| n.toPrecision(4) | =>1.235e+5 |
| n.toPrecision(7) | =>12345.8 |
| n.toPrecision(10) | =>123456.7890 |

parseInt()可以接受第二个可选参数，这个参数指定数字转换的基数，合法的取值范围是2-36

|  |  |
| --- | --- |
| parseInt(“3 weev”) | 3 |
| parseInt(“-12.34”) | -12 |
| parseInt(“oxFF”) | 255 |
| parseInt(“-oxFF”) | -255 |
| parseFloat(“.1”) | O.1 |
| parseInt(“o.1”) | 0 |
| parseInt(“.1”) | NaN |
| parseFloat(“$72.47”); | NaN |
| parseInt(“11”,2); | 3 |
| parseInt(“007”,10) | 77 |

3.8.2对象转换为原始值

对象到字符串和对象到数字的转换是通过调用待转对象的一个方法来完成的，值得注意的是，这里提到的字符串和数字的转换规则只适用本地对象。宿主对象根据各自的算法可以转换成字符串和数字。

所有的对象继承了两个转换方法。第一个是tostring（），它的作用是返回这个对象的字符串

日期类定义的toString（）方法返回了一个可读的日期和时间字符串。RegExp 类定义的tostring（）方法将RegExp对象转换为表示正则表达式直接量的字符串。

日期定义的valueof()方法会返回它的一个内部表示：1970，1，1以来的毫米数

Var d=new Date（2012，0，1) d.valueof()

**3.10.3作用域链**

当定义一个函数时，它实际上保存一个作用域链。当调用这个函数时，它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的链

**第四章 表达式和运算符**

**4.4属性访问表达式**

两种形式：expression.identifier

expressoin[expression]

Eg\*

|  |  |
| --- | --- |
| Var 0={x:1,y:{z:3}}; | 一个实例对象 |
| Var a=[0,4,[5,6]]; | 一个包含这个对象的实力数组 |
| 0.x | =>1:表达式0的x属性 |
| 0.y.z | =>3:表达式0.y的z属性 |
| 0[“x”] | =>1:对象0的x属性 |
| a[1] | =>4:表达式a中索引为1的元素 |
| a[2][“1”] | =>6:表达式a[2]中索引为1的元素 |
| a[0].x | =>1:表达式a[0]的x属性 |

如果属性名称是一个保留字或者包含空格和标点符号，或者是一个数字（对于数组来说）则必须使用方括号的形式，当属性名是通过计算得出的，则要采用方括号的形式。

**4.5调用表达式**

当对调用表达式进行求值时，首先计算函数表达式，然后计算参数表达式，得到一组参数值，然后，实参的值将一次赋值给形参，这些形参是定义函数时指定的，接下来开始执行函数体，如果函数使用return函数并有一个值，则这个返回值就是整个调用函数表达式的值。否则为undefined，

**4.6对象创建表达式**

对象创建表达式创建一个对象，并调用一个函数，（这个函数成为构造函数）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前多了一个关键字new。

**4.7.3左值**

左值是指表达式只能出现在赋值运算符的左边，变量，数组元素，对象属性均为左值，允许内置函数返回一个左值，但自定义的函数则不能返回。

**4.8.1“+”运算符**

加法操作符的行为表现

如果其中一个操作数是对象，则会转换为原始值

|  |  |
| --- | --- |
| 1+2 | 3 |
| “1”+”2” | 12 |
| “1”+2 | 12 |
| 1+{} | 1[object Object] |
| True+true | 2 |
| 2+null | 2 |
| 2+undefined | NaN |
| 1+2+”qz” | 3qz |
| 1+(2+”qz”) | 12qz |

**4.8.2一员算数运算符**

Var i=1，j=++i;//i和j都为2

Var i=1, j=i++;//i为2 j为1

**4.9关系表达式**

对象的比较是引用的比较，而不是值的比较，对象和其本身是想等的，和其他任何对象都不相等，

“11”<”3” true

**4.9.2比较运算符**

对于数字和字符串的操作符来说，加号运算符和比较运算符有所不同，前者更偏爱字符串。

**4.9.3in运算符**

|  |  |
| --- | --- |
| Var data=[7,8,9]; |  |
| “0”in data | =>true//数组包含元素“0” |
| 1 in data | =>true//数字转换为字符串 |
| 3 in data | =>false//没有索引为3的元素 |

**4.9.4 instanceof运算符**

需要注意的是 所有对象都是object实例，当通过instanceof判断一个对象是否是一个实例的时候，这个对象也会包含对父类的检测，如果instance of左操作数不是对象的话，instance of返回false，如果右操作数不是函数的话，将会返回类型错误异常。

为了计算0 instanceof f，首先计算f.prototype,然后在原形链中查找o,如果找到，那么0

是f的一个实例，或者是f的父类的实例，表达式返回true，如果f.prototype不在0的原形链上，那么0就不是f的一个实例。返回false。

**4.10.1 逻辑&&**

假值是false null undefined “” 0 -0 NaN 其他为真值，

**4.11赋值表达式**

复制操作符的结合性从右向左，i=j=k=0；

**4.13.1条件运算符**

X>0?x:-x//x的绝对值

**4.13.2 type of**

所有可执行对象进行typeof运算都将返回function，非原生可执行对象的typeof运算将返回object

**4.13.2delete运算符**

它用来删除对象属性和数组元素，是用来做删除工作的，不是用来返回一个值的，删除数组元素后数组长度不变，通过function语句定义的函数和函数参数也不能删除。

|  |  |
| --- | --- |
| Var o={X:1,Y:2}; | //定义一个对象，初始化为对象 |
| delete 0.x | //删除一个对象属性 返回为true |
| typeof 0.x | 属性不存在，返回undefined |
| delete 0.x | 删除不存在的属性，返回为true |
| delete 0 | 不能删除通过var声明的变量，返回false |
|  | 在严格模式下，将抛出一个异常 |
| Delete1 | 参数不是一个左值，返回true |
| this.x=1 | 给全局变量定义一个属性，这里没有使用var |
| Deletex | 试图删除它，返回true |
|  | 在严格模式下会抛出异常这时使用deletethis.x |
| x； | 运行错误，没有定义x |

1. **语句**

跳转语句：可以让解释器跳转至程序的其他部分执行，如break，return，throw

5.5循环

5.5.1 while

输出0-9

Var count =0;

While(count<10)

{document.write(count);

Count++;}

5.5.3 for

For(var i=0;i<10;i++){

Document.write(i)} 0-9

5.5.4for in

For/in并不会遍历对象的所有的属性，只有可枚举的属性才会遍历得到，javascript语言核心所定义的内置方法就不是可枚举的。所有的对象都有tostring（）方法，但for/in循环并不枚举tostring（）属性

5.6.2break语句

单独使用break语句的作用立即退出最内层的循环，当循环终止条件非常复杂时，在函数中使用break语句实现这些条件判断的做法要比直接在循环表达式中写出这个在复杂终止条件的做法简单很多。

与要注意的是，不管break语句带不带标签，它的控制权都无法越过函数的边界，比如，对于一条带标签的函数定义语句来说，不能从函数内部通过这个标签来跳转到函数外部。

5.6.3continue语句

Continue转而执行下一个循环

5.6.4return语句

Return语句只能出现在函数体内，

function square(x){

return x\*x;

}

square(2)

如果没有return语句，则函数调用仅依次执行函数体内的每一条语句直到函数结束，最后返回调用程序，return语句经常作为最后一条语句出现，return语句单独使用，调用表达式的结果是undefined。

1. 对象

6.1.4 Object.create()

它创建了一个新对象，其中第一个参数是这个对象的原型，第二个可选参数，用以对对象的属性的描述

6.3删除属性

|  |  |
| --- | --- |
| O={x:1}; | O 有一个属性x并继承tostring属性 |
| Delete o.x; | 删除x，返回true |
| Delete o.x; | x已经不存在了，返回true |
| Delete o.tostring; | Tostring是继承来的，返回true |
| Delete 1 | 无意义，返回true |

Delete不能删除那些可配置性味false的属性。

6.4检测属性

除了使用in运算符外，另一种更简便的方法是使用！==判断一个属性是否是undefined，然而有一种场景只能使用in运算符，in可以区分不存在的属性和存在但值为undefined的属性

|  |  |
| --- | --- |
| Var o={x:undefined}; | 属性被显示赋值为undefined |
| O.x!==undefined | False：属性存在，但值为undefined |
| O.y!==undefined | False：属性不存在 |
| “o”in x; | True：属性存在 |
| “o”in y; | False：属性不存在 |
| Delete o.x; | 删除了属性x |
| “x”in o | False：属性不再存在 |

6.8对象的三个属性

每个对象都有与之相关的原型，类，和可拓展性。

6.8.1原型属性

经常使用表达式o.constructor.prototype来检测一个对象的原型。要想检测一个对象是否是另一个对象的原型或处于原型链中，请使用isPrototypeof()方法，例如，可以通过p.isProrotypeOf(o)来检测p是否是o的原型，需要注意的是isPrototypeof()方法和instanceof()

运算符非常类似。

\_\_proto\_\_属性查询原型，safari chrome支持，IE opera不支持。

6.10.4valueof

Valueof()和tostring()方法十分类似，需要将对象转化为原始值而非字符串的时候才会调用它，尤其是转化为字符串的时候。如果在需要原始值的上下文使用了对象，javascript自会调用这个方法。

1. **数组**

7.1创建数组

数组直接量的值不一定要是常量，它们可以是任意的表达式，它可以包含对象直接量和数组直接量，如果省略数组元素的任一值，省略的元素将被赋予undefined。

数组是对象的特殊形式，使用方括号访问数组元素就像方括号访问对象一样。

7.2数组元素的读和写

注意：可以使用负数或非整数来索引数组，这种情况下，数组转化为字符串，字符串作为属性名来使用，既然名字不是非负整数，它就被当作正常常规的对象属性，而非数组的索引，

同样，如果凑巧使用了是飞负整数的字符串，它就当作数组索引，而非对象属性，当使用的一个浮点数和一个整数相等时情况也是这样的。数组没有“越界” 的概念，当试图查询对象中不存在的属性时，不会报错，只会得到undefined值，类似于对象。

**7.4数组长度**

1如果为一个数组的元素赋值，它的索引i大于或等于现有数组的长度时，length属性的值将设置为i+1。

2设置length属性为一个小于当前长度的非负整数n时，当前数组中那些索引值大于或等于n的元素将从中删除。

7.6数组遍历

在嵌套循环或其他性能非常重要的上下文中，可以看到这种基本的数组遍历需要优化，数组的长度应该只查询一次而非每次循环都要查询

for(var i=0;i<a.length;i++){

if(a[i])continue;//跳过null undefined和不存在的元素

//循环体

}

for(var i=0;i<a.length;i++){

if(a[i]===undefined)continue;//跳过 undefined和不存在的元素

//循环体

}

for(var i=0;i<a.length;i++){

if(！i in a)continue;//跳过不存在的元素

//循环体

}

允许for/in循环以不同的顺序遍历对象的属性，通常数组元素的遍历是升序的，如果算法依赖于便利的顺序，那么最好不要使用for/in而是常规的for

**7.9数组方法**

在大多数情况下，调用提供的函数使用三个参数：数组元素，元素的索引，和数组本身，通常，只需要第一个函数值，可以忽略后两个参数，数组方法的第一个参数是一个函数，第二个参数是可选的，如果有第二个参数则调用的函数被看作是第二个参数的方法，也就是说，在调用函数时传递进去的第二个参数作为它的this的关键值来使用，被调用的函数的返回值非常重要，但是不同的方法处理返回值的方式也不一样，数组方法都不会修改它们调用的原始数组，当然传递给这些方法的函数可以修改这些数组。

7.10数组类型

数组是具有特殊行为的对象，给定一个未知的对象，判定它是否为数组通常非常有用，可以使用Array.isArray()函数来做这件事，

Array.isArray([])==>true

Array.isArray({})==>false

7.11类数组对象

#当有新的元素添加到列表中时，自动更新length属性

#设置length为一个较小值将截断数组

#从Array.prototype中继承一些有用的方法

#其类属性为Array

以下代码为一个常规对象增加了一些属性，使其变成类数组对象，然后遍历生成的伪数组的元素

Array.isArray([])==>true

Array.isArray({})==>false

var a={};//从一个常规空对象开始

//添加一些属性，成为“类数组”

var i=0;

while(i<10){

a[i]=i\*i;

i++;

}

1. length=i;
2. 当作真正的数组便利它

var total=0;

for(var j=0;j<a.length;j++)

total+=a[j];

7.12作为数组的字符串

字符串的行为类似于只读的数组，除了用charAt()方法来访问单个的字符意外，还可以使用单括号。

var s=test;

s.charat(0)//=>"t"

s[1]//=>"e"

\*字符串是不可变值，故把他们当作数组看待时，它们是只读的，如push()sort()reverse()splice()等数组方法会修改数组，它们在数组上是无效的。不仅如此，使用数组方法来修改字符串会导致错误，出错的时候没有提示。

1. 函数

Return语句导致函数停止执行，并返回它的它的表达式的值给调用者，如果return语句没有一个与之相关的函数表达式，则它返回undefined

8.2函数调用

构成函数主体的代码在定义之时并不会执行，只有定义该函数时，它们才会执行，有四种方法来调用函数

#作为函数

#作为方法

#作为构造函数

#通过它们的call()和apply()方法间接调用

使用调用表达式可以进行普通的函数调用，也可以进行方法调用，一个调用表达式由多个函数表达式组成，如果函数表达式是一个属性访问表达式，即该函数是一个对象的属性或数组中的一个元素，那么他就是一个方法调用表达式。

在一个调用中每个参数表达式都会计算出一个值，计算的结果作为参数传递给另一个函数，

对于普通的函数调用，函数的返回值成为调用表达式的值。

8.2.2方法调用

需要注意的是，this是一个关键字不是变量，也不是属性名语法不允许给this赋值，关键字this没有作用域的限制，嵌套的函数不会从它的调用函数中继承this

8.6闭包

函数对象可以通过作用域链相互关联起来，函数体内部的变量都可以保存在函数作用域内，这种特性成为闭包。

var s ="g";

function c(){

var s ="l s";

function f(){return s;}

return f();

}

C()

function counter(){

var n=0;

return {

count :function(){return n++;},

reset:function(){n=0;};

}

var c=counter(),d=counter();//创建两个计数器

c.count()//==>0

d.count()//==>0它们互补干扰

c.reset()//reset()和count()方法共享状态

c.count()//==>因为重置了c

d.count()//==>而没有重置d

}