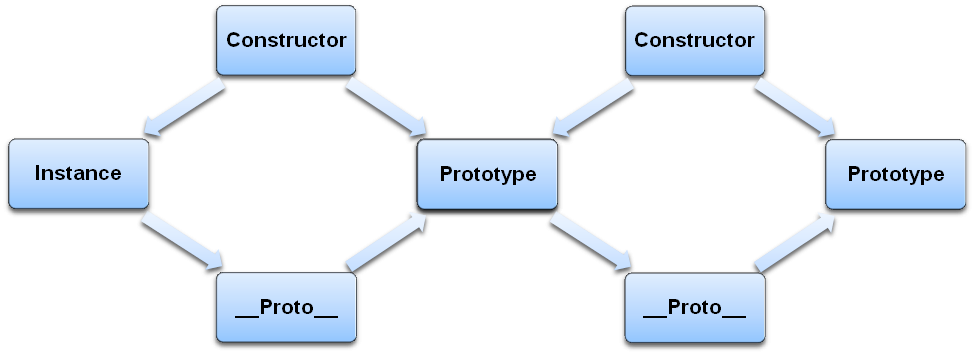
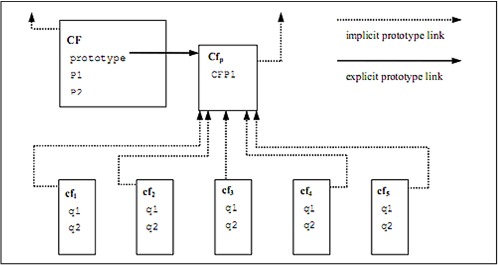
**ECMA262深入浅出**

ECMA-262简述  
ECMAScript语言概述  
ECMAScript对象概述(原型链Prototype Chain)  
ECMA-262主要术语  
ECMAScript执行环境(作用域链Scope Chain，闭包机制)  
ECMAScript函数(new原理)  
ECMAScript内部属性(参考)  
ECMAScript执行环境作用域链图示  
ECMA-262参考资料  
  
**ECMA-262简述**  
  
ECMA(European Computer Manufacturers Association)  
  
ECMA-262标准的最初的版本是基于Brendan Eich创造的JavaScript语言来制定的，它是ECMAScript脚本语言的规范及标准。该标准在1998年4 月通过快速通道提交到ISO/IEC JTC 1并被采纳为国际标准ISO/IEC 16262，所以之后一直简称为ECMA-262，该标准发展到现在已经到了第5版。   
  
  
所有遵循ECMA-262标准实现的脚本都可以称之为ECMAScript，如JavaScript, JScript, ActionScript等等。  
  
注意：ECMA-262是从ECMAScript脚本引擎实现的角度去描述的。  
  
我们目前所使用的大部分浏览器中的脚本引擎多数都是基于ECMA-262标准第3版的实现，参见下图：  
  
  
  
**ECMAScript语言概述**  
  
ECMAScript最初的设计是一种网页脚本语言。  
  
  
脚本语言是一种编程语言，可用来操作、自定义、自动化现有系统中的设备。  
在某些系统中，一些可用的功能是可以通过用户接口来使用的。而脚本语言，就是将这些功能暴露给编程人员进行控制的机制。这样的系统，我们可以说，它们为我们提供了一个运行脚本语言的宿主环境。脚本语言是专门给专业和非专业的编程人员使用的，为了适应非专业的程序员，有些方面多少有些不太严格。  
  
脚本语言与宿主环境是两个不同的主体，比如，浏览器就为ECMAScript提供了一个宿主环境，BOM和DOM则都是浏览器提供的功能，而各种基于ECMA262实现的脚本语言，如JavaScript,Jscript等，都可以为我们提供对BOM和DOM的操作方法。因为宿主环境的不同，它们提供的功能也不尽相同，这也是我们知道的，不同的浏览器中会有不同的BOM和DOM的属性和方法。  
  
  
ECMAScript是基于对象(object-based)的编程语言，在宿主环境中执行计算或操作对象。ECMAScript程序就是一个对象间交互通信的聚合。 在ECMAScript中，“对象”(object)的properties是一个无序的集合，properties 是一个容器，可以包含其他的对象，原始值(primitive values)或方法(methods)。  
  
原始值属于下列“内置类型”(built-in types)中的一员：Undefined, Null, Boolean, Number和String,“对象 ”则是内置类型：Object,“方法”是一个透过property关联到对象的函数。  
  
  
ECMAScript定义了“内置对象”(built-in objects)，以使ECMAScript实体更为完善。这些内置对象包括：Global对象，Object对象， Function对象，Array对象，String对象，Boolean对象，Number对象，Math对象，Date对象， RegExp对象和错误对象： Error,EvalError,RangeError,ReferenceError,SyntaxError,TypeError,URIError。  
  
  
ECMAScript还定义了“内置操作符”或者说是“函数”或“方法”。ECMAScript操作符包括了各种一元运算符、乘除运算符、加减法运算符、位移运算符、关系运算符、相等运算符、二元运算符、二元逻辑运算符、分配运算符和逗号运算符。  
  
  
ECMAScript的语法类似Java的语法。ECMAScript语法宽松是为了使它能够成为更容易使用的脚本语言。比如，变量不需要申明类型也不需要为properties指定类型，函数定义也可以不需要出现在它们的调用语句之前。  
  
**ECMAScript对象概述**  
  
ECMAScript没有像C++，Smalltalk，Java等语言中的真正的类，但是它支持构造器(constructors)。它在代码执行时创建对象，为对象分配内存并初始化它们全部或部分初始值和properties。所有的构造器都是对象，但不是所有对象都是构造器。每个构造器都有一个Prototype属性用于实现基于原型的继承(prototype-based inheritance)和属性共享(shared properties)。  
  
对象在对构造器使用new表达式时被创建，比如，new String(“A String”)创建了一个新的字符串对象。不用new表达式直接去调用一个构造器，其结果将依赖于这个构造器的具体实现，如，String("A String") 产生一个原始值String，而不是一个对象。  
  
  
ECMAScript支持基于原型的继承。所有构造器都有一个相关联的原型，所有由构造器创建的对象都隐含一个指向到该原型的引用（称为对象原型），此外，一个原型可能有一个非空的、隐含的引用，指向到它的原型，所以，这又被称为是：原型链(prototype chain)  
  
  
  
当一个引用来自一个对象时，它会去该对象和它的原型(链)中查找与该property同名的属性，换句话说，会先直接从该对象中检测是否存在这个同名的引用，如果有则返回，否则再从该对象的原型(链)中去检测是否存在该引用。(DEMO)  
  
DEMO:  
<script type="text/javascript">  
Object.prototype.hi = function(){  
    alert('object.prototype.hi');  
};  
Object.prototype.greetings = function(){  
    alert('object.prototype.greetings');  
};  
  
Function.prototype.greetings = function(){  
    alert('function.prototype.greetings');  
};  
  
var CF = function(){};  
CF.prototype.hello = function(){  
    alert('CF.prototype.hello');  
};  
CF.prototype.bye = function(){  
    alert('CF.prototype.bye');  
};  
  
var o = new CF();  
o.hello = function(){alert('o.hello');};  
  
o.hello();  
o.bye();  
o.hi();  
o.greetings();  
  
CF.hi();  
CF.greetings();  
</script>  
  
  
在基于类的面向对象语言里,通常,状态由实例持有，方法由类持有，继承只有结构和行为。在ECMAScript中，状态和方法由对象持有，结构、行为和状态都可以被继承。  
  
对象通常都不会直接包含那些由原型包含共享的属性和值。描述见下图：  
  
  
  
CF是一个构造函数（同时也是一个对象）。通过对它进行new表达式创建了5个实例对象：cf1,cf2,cf3,cf4,cf5。每个实例对象都包含有q1 和q2两个属性。(DEMO)  
  
例如，Cfp是cf3的原型(虚线，表示隐含的原型关系)，构造函数CF自己有两个属性P1和P2，它们对 Cfp,cf1,cf2,cf3,cf4,cf5都不可见。Cfp的属性CFP1对cf1,cf2,cf3,cf4,cf5可见（但不包括CF），还有那些在Cfp的隐含的原型链中找到的 property name(非q1,q2或者CFP1的其他属性)。注意，CF和Cfp之间并非隐含的原型联系。  
与基于类的对象语言不同，ECMAScript对象的属性可以通过给对象赋值的方式动态地添加到对象上，意思就是，构造函数不需要一个个地为那些由它构造出的对象赋值，如上图，只要为CFp添加一个属性值,即可使得由CF构造出的所有对象：cf1,cf2,cf3,cf4,cf5共享。  
  
DEMO:  
<script type="text/javascript">  
var CF = function(){};  
CF.p1 = 'p1';  
CF.p2 = 'p2';  
  
CF.prototype = {CFP1:'cfp1'};  
  
var cf1 = new CF(),cf2 = new CF(),cf3 = new CF(),cf4 = new CF(),cf5 = new CF();  
  
alert(cf1.p1+','+cf2.p2);//构造函数自身的属性对由它构造出的对象不可见  
  
alert(CF.CFP1);//对CF不可见  
alert(cf1.CFP1);//对CF构造的所有对象可见  
  
cf3.\_\_proto\_\_.CFP2 = 'cfp2';//在构造函数原型上添加属性  
//\_\_proto\_\_相当于从cf3到CF.prototype的隐含的原型关系，该属性是SpiderMonkey引擎的私有属性，仅限在SpiderMonkey引擎中访问  
  
alert(cf1.CFP2);//即可对所有该构造函数构造的对象进行共享  
</script>  
  
**ECMA-262主要术语**  
  
以下是ECMAScript中主要术语的非正式定义。  
  
Type  
数据值的集合  
Primitive Value  
原始值包括Undefined,Null,Boolean,Number,String这些类型成员。原始值是语言实现中底层可以直接表示的数据。  
Object  
每个对象都是Object类型的成员。它包含的所有属性成员，如原始值、对象或者函数，是一个无序的集合。对象属性成员放置的函数又被称为方法。  
Constructor  
构造器是一个函数对象，用来创建和初始化对象。每个构造器都有一个关联的原型对象用来实现属性成员的继承和共享。  
  
Prototype   
ECMAScript中的原型用来实现对象结构、状态和行为的继承。当一个构造器构造了一个对象，这个对象会隐含地引用到该构造器关联的原型去解析属性的引用。构造器关联的原型可以通过程序表达式constructor.prototype 来引用。在某个共享对象的原型上添加的属性值，通过继承，可以被所有其他对象共享。  
Native Object   
原生对象是由ECMAScript的实现提供的，独立于宿主环境。有些原生对像同时又是内置对象，其他的则可能会在执行一个 ECMAScript程序时产生。  
Built-in Object   
内置对象是由ECMAScript的实现提供的，独立于宿主环境，它们出现在ECMAScript程序开始执行的时候。所有的内置对象都是原生对象。  
Host Object  
宿主对象由为ECMAScript提供完整执行环境的宿主环境提供的。任何对象，不是原生对象，就是宿主对象。  
  
Undefined Value  
Undefined值是一个原始值，当一个变量没有被分配值的时候使用。  
Undefined Type  
Undefined类型只有一个值，undefined。  
Null Value  
Null值是一个原始值，它表示空的，没有的，或不存在的引用。  
Null Type  
Null类型有一个值，null。  
Boolean Value  
Boolean值是Boolean类型的成员，它只能是两个值中的一个，true或false。  
Boolean Type  
Boolean类型表示一个逻辑实体，由两个唯一值组成，一个是true，另一个是false。  
  
Boolean Object   
Boolean对象是Object类型的成员，它是内置对象Boolean的实例。意思是，一个Boolean对象是通过Boolean构造器的 new表 达式创建，提供一个boolean参数。结果对象有一个隐含的（未命名）原型boolean。一个Boolean对象支配一个 Boolean值。  
String Value   
String值是String类型的成员，它是一个有限的有序的，0到16位无符号整数长度的值。注意：尽管每个值通常都表示一个单独的16位的UTF-16文本的单元，但该语言不会做出任何限制和要求当这个值不是16位无符号整数的情况时。  
String Type   
String类型是所有字符串值的集合。  
String Object   
String对象是Object类型的成员，它是内置对象String的实例。  
  
Number Value   
Number值是Number类型的成员，它是一个直接表示的数字。  
Number Type   
Number类型是表示数字的值的集合。在ECMAScript中，该集合表示IEEE754的64位双精度运算格式的值，包含一个特殊值"Not-a- Number"(NaN)，正无穷大和负无穷大。  
Number Object   
Number对象是Object类型的成员，它是内置对象Number的实例。  
Infinity   
原始值Infinity表示一个正无穷大的数字，Number类型成员。  
NaN  
原始值NaN表示IEEE标准"Not-a-Number"值的集合，Number类型成员。

**ECMAScript执行环境**  
  
什么是执行环境?  
  
想象一下，假设你就是一个ECMAScript引擎。  
  
当你接收到一条语句：  
  
this.x = y;  
  
你能确定该做什么吗？  
  
很明显， 我们不知道该做些什么，因为我们既不知道this是什么，也不清楚y是什么。  
  
所以，如果要正确执行这条语句，我们需要一个上下文，用以确定this和y到底是什么，这个上下文，就是执行环境。  
  
**执行环境**  
  
当控制器转到ECMAScript可执行代码时，即会进入到一个执行环境中。活动的执行环境会在逻辑上形成一个堆栈，在这个堆栈的顶端存放的是运行时的执行环境。  
  
**函数对象**(两种类型)  
通过在源文件中透过函数声明、函数表达式或内置Function对象来定义的函数对象。  
内部函数对象，它们属于内置对象，如parseInt, Math.exp等等。  
  
**可执行代码** (三种ECMAScript可执行代码类型)  
Global代码  
Eval代码  
Function代码  
  
**变量实例**  
  
每个执行环境都会与一个“变量对象”关联。在源文件中定义的的变量和函数都会作为properties添加到该对象中去。例如函数的代码，参数将作为properties添加到该函数执行环境的变量对象中去。  
  
  
**在进入一个执行环境时，“变量对象”绑定properties的顺序如下**：(DEMO)  
  
**针对函数代码**：为每一个在参数列表中定义的形参，创建一个同名的标识符作为property添加到“变量对象”中，其值由调用[[call]]时以参数的形式提供。如果调用者提供的参数值少于形参数量，其他的形参值将为undefined。如果遇到多个同名的形参，最后一个同名形参的值将被保留，如果这最后一个形参也未被提供值，同样会是undefined。  
  
**针对函数声明**：在“变量对象”中创建一个以它们的名字作为标识符的property，其值为创建后的函数对象。如果“变量对象”中已经有一个同名的property，则将它替换掉。  
  
**针对变量声明**：在“变量对象”中创建一个以它们的名字作为标识符的property，其值为undefined，如果之前已有这个property,则其值不变。  
  
DEMO:  
<script type="text/javascript">  
/\*-------------函数代码------------\*/  
function x(a,a,b){alert(a);alert(b);};  
x(1,2,3);  
x(1);  
/\*-------------函数声明------------\*/  
function a(){alert(1);};  
alert(a);  
function a(){alert(2)};  
alert(a);  
/\*-------------变量声明------------\*/  
var b;  
alert(b); //其值为undefined  
</script>  
  
  
**作用域链与标识符查找**  
  
所有的执行环境都与一个作用域链(scope chain)相关联。作用域链是一个用来查找标识符的对象列表。当控制器进入到一个执行环境中的时候，作用域链被创建并放置了一个初始的对象集合。在一个执行环境运行的时候，它的作用域链只受with语句和catch子句影响。在执行的时候，标识符查找的方法如下：  
  
1. 获取Scope Chain的下一个对象。如果没有对象了，则转到第5步   
2. 调用Result(1)的[[HasProperty]]方法, 传递Identifier作为参数   
3. 如果Result(2)是true, 则返回一个Reference(引用)类型的值，它的base object是Result(1)，而它的property name是Identifier  
4. 跳到第1步  
5. 返回一个Reference类型，它的base object是null，它的property name是Identifier  
  
标识符查找的结果通常是一个名字为标识符字符串的引用值。  
  
  
**全局对象(Global Object)**  
  
在控制器进入任何执行环境之前，会创建一个唯一的全局对象，初始化的全局对象包括以下属性：  
  
**内置对象**，如 Math,String,Date,parseInt等等，它们都是不可枚举的(DontEnum)。  
  
**宿主属性**，它自己可能会包含一个值为全局对象的属性，比如，在HTML 的DOM中，全局对象中的window属性，就是全局对象自己。  
  
当控制器进入了执行环境，ECMAScript运行的时候，全局对象可能还会被添加上一些额外的属性，一些之前已有的属性也有可能被改变。  
  
**This**  
  
所有活动的执行环境都会有一个this值，它依赖并取决于代码执行时的caller。  
  
同一个执行环境中的this值是不变的。  
  
  
**Arguments对象**  
  
当控制器进入到一个函数的执行环境中时，会创建一个arguments对象,初始过程如下：  
  
arguments对象的内部属性[[prototype]]，是原生对象prototype，初始值是Object.prototype  
  
创建callee属性，不可枚举(DontEnum)，其值为正在执行的函数对象，这使得匿名函数也可以进行递归。  
  
创建length属性，不可枚举(DontEnum)，其值为caller提供的实际参数数量。  
  
按少于length的正整数数量创建ToString(arg)属性，不可枚举(DontEnum)，第一个实际参数值对应为arg = 0，第二个对应为arg = 1，依此类推。当实际参数少于形参时，该属性会在活动对象中相应地共享它的值。意思就是说，改变此属性将改变活动对象中的属性值，反之亦然。  
  
  
**进入一个执行环境时发生的那些事儿**  
  
每次调用函数或构造器都会进入一个新的执行环境，即使在一个函数递归地调用自己的时候亦是如此。每次return，都会退出这个执行环境。一个未捕捉的异常也可能退出一个或多个执行环境。  
  
当控制器进入到一个执行环境中时，作用域链即被创建和初始化，变量实例化也开始进行，this值也被确定下来。  
  
**Global代码**  
  
作用域链创建并初始，仅包含global对象。  
  
变量实例化进行的时候以global对象作为变量对象，属性为不可删除(DontDelete) 。  
  
This值为global对象。  
  
**Eval代码**  
  
当控制器进入到eval代码的执行环境中时，先前活动的执行环境，引用到调用环境\*，用来决定作用域链，变量对象和this值。如果没有调用环境，则所有这些都以全局代码形式进行处理。  
  
作用域链初始化时包含一个与调用环境相同的对象，并保持相同的顺序。这包含了通过with语句和catch子句添加到调用环境作用域链上的对象。  
  
变量实例化时使用调用环境下的变量对象。  
  
This值与调用环境下的this值一致。  
  
\* 调用函数语句执行时的execution context就是calling context  
  
**Function代码**  
  
作用域链初始化时先添加活动对象，然后再添加该函数对象[[Scope]]属性中存储的其他对象。\* 闭包机制  
  
变量实例化执行时使用活动对象作为变量对象，属性为不可删除(DontDelete)。  
  
This的值为Caller，如果caller不是对象(或者是null)，则this值为global对象。  
  
  
**With语句**  
  
with语句会在当前执行环境中的作用域链顶端添加一个计算对象，在这个扩展的作用域链执行完语句之后，随机复原之前的作用域链。  
  
With(表达式)语句的执行过程如下：  
  
1.评估表达式  
2.调用GetValue(Result(1))  
3.调用ToObject(Result(2))  
4.将Result(3)添加到 作用域链的顶端  
5.用第4步扩展的作用域链来评估语句  
6.使C=Result(5)，如果在第5步时有异常抛出，则C=(throw,V,empty)，V是这个异常（执行现在当作没有异常抛出继续进行）  
7.在作用域链中移除Result(3)  
8.返回C  
  
注意：当控制器离开该“嵌入”的语句时，无论是正常还是异常，作用域链都将被恢复。  
  
  
**Catch子句**  
  
catch(标识符)块的执行过程如下：  
  
1.使C为通过catch传递进来的参数  
2.像new Object()一样创建一个新的对象  
3.在Result(2) 的那个对象上创建一个property，名字是标识符，值是C，不可删除  
4.将Result(2)添加到作用域链的顶端  
5.评估这个块语句  
6.将Result(2) 从作用域链中移除  
7.返回Result(5)  
  
**函数声明**  
  
function Identifier(FormalParameterList opt){FunctionBody}  
  
函数声明的解析过程如下：  
  
1.创建一个new Function对象， FormalParameterList指定参数，FunctionBody指定函数体。将当前正在运行的执行环境中的作用域链作为它的作用域。  
  
2.为当前的变量对象 创建一个名为Identifier的属性，值为Result(1)。  
  
  
**匿名函数表达式**  
  
function(FormalParameterList opt){FunctionBody}  
  
匿名函数表达式的解析过程如下：  
  
1.创建一个new Function对象， FormalParameterList指定参数，FunctionBody指定函数体。将当前正在运行的执行环境中的作用域链作为它的作用域。  
  
2.返回Result(1)。  
  
**具名函数表达式**  
  
function Identifier(FormalParameterList opt){FunctionBody}  
  
具名函数表达式的解析过程如下：  
  
1.创建一个new Object对象  
2.将Result(1)添加到作用域链的顶端  
3.创建一个new Function对象，FormalParameterList指定参数，FunctionBody指定函数体。将当前正在运行的执行环境中的作用域链作为它的作用域。  
4.为Result(1)创建一个名为Identifier的属性，其值为Result(3)，只读，不可删除  
5.从作用域链中移除Result(1)  
6.返回Result(3)  
  
**创建函数对象**  
  
函数对象构造过程如下：  
  
1.如果已有对象E，它已有FunctionBody，且与现在给到的FunctionBody相等，则跳转到第13步  
2. 创建一个新的ECMAScript native对象F  
3.设置F的[[Class]]为“Function”  
4.设置F的[[Prototype]]为原始的Function prototype对象  
5.设 置F的[[Call]](参见之后的Slide)  
6.设置F的[[Construct]]（参见之后的Slide）  
7.设置F的[[Scope]]为一个新的作用域链，它和当前作用域包含相同的对象  
8.设置F的lengeh属性值为形参数量，如果未定义参数，则为0。9.以new Object()表达式方式创建一个新的对象  
10.设置F的 constructor属性值为Result(9)  
11.设置F的prototype属性值为Result(9)  
12.返回F  
13.酌情（由实现者决定）跳转到第2步或第14步  
14.创建 一个新的ECMAScript native对象F，连接到 E，复制所有E和F的非内部属性\*以使得它们与E和F的保持一致。  
15.设置F的[[Class]]属性值为“ Function”  
16.设置F的[[Prototype]]属性值为原始的Function prototype对象  
17.设置F的[[Call]]属性  
18.设置F的[[Construct]]属性  
19. 设置F的[[Scope]]属性值为一个新的作用域链，它和当前作用域包含相同的对象  
20.返回F  
  
\* 内部属性指的是所有以[[]]书写的属性  
  
  
**创建函数对象**  
  
注意：  
  
每个function都会自动创建一个prototype属性，以使得它们可以当作构造器来使用。  
  
第1步，使得类似函数A中嵌套了一个不依赖于A的函数B的这种情况进行优化。在这种情形的实现中，允许在A每次被调用的时候，重用B这个对象。  
  
第13步决定是否执行此 优化。如果在某个实现中选择不，则会跳转到第2步。  
  
  
**创建函数对象**  
  
连接对象(很多引擎都未去做这个实现)   
  
当两个或更多的Function对象连接时，它们具有如下特殊的行为：  
  
1.任何时候，对 象O创建或设置一个非内部属性时，通信机制会立即在其他所有与O连接的对象上做相同的操作。  
  
2.任何时候，对象O删除一个非内部属性， 通信机制会立即在其他所有与O连接的对象上做相同的操作。  
  
3.如果对象O与P相连，它们可以通过==和===来进行比较。  
  
4.当对象O与P相连 ，且对象P与Q相连，则O与Q也会自动相连。  
  
注意：相连的对象除了它们各自不同的内部属性外，是很难进行区别的。而这些内部属性可能也只有[[Scope]]会不同。  
  
**调用函数对象**  
  
[[Call]]  
  
当函数对象F调用了[[Call]]属性，将执行以下步骤：  
  
1.将F的形参表、传递的参数列表，和this植入到一个新的执行环境  
  
2.评估F的FunctionBody  
  
3.退出在第1步植入的执行环境，恢复之前的执行环境  
  
4.如果Result(2).type是 throw,抛出Result (2).value  
  
5.如果Result(2).type是return，返回Result(2).value  
  
6.Result(2).type默认情况下，返回undefined  
  
  
**new操作原理**  
  
[[Construct]]  
  
当函数对象F调用了[[Construct]]属性，将执行以下步骤：  
  
1.创建一个新的ECMAScript native对象  
2.将 Result(1)的[[Class]]属性值设为“Object”  
3.取得F的prototype属性值  
4.如果Result(3)是一个对象，将Result(1)的[[Prototype]]属性值设置为Result(3)  
5.如果Result(3)不是一个对象，将Result(1)的[[Prototype]]属性值设置为原始的Object prototype对象  
6.调用F的 [[Call]]属性，将Result(1)设置为this值，将[[Construct]]传递的参数当作参数列表  
7.如果Type(Result(6))是一个对象，返回Result(6)   
8.返回Result(1)  
  
DEMO:  
<script type="text/javascript">  
/\*new操作原理（spiderMonkey引擎下测试）\*/  
  
var a = function(sA,sH){  
    var x = "x";  
    this.a = sA;  
    this.h = sH;  
    this.say = function(){alert(this.a+','+x)}  
}  
a.prototype.hi = function(){alert(this.h)}  
  
var createInstance = function(source){  
    var p = {}  
    var args = Array.prototype.slice.call(arguments,1);  
    p.\_\_proto\_\_ = source.prototype;  
    source.apply(p,args);  
    return p;  
}  
  
var A = createInstance(a,"A","hi A");  
A.say();  
A.hi();  
</script>  
  
**ECMAScript内部属性**  
  
[[Prototype]]  
This对象的原型，该属性值只会是一个object或null，所有的[[Prototype]] chain最终都都会通向到null。  
[[Class]]  
指明This对象类别的字符串值  
[[Value]]  
与This对象关联的内部状态信息  
[[Get]](PropertyName)  
返回指定属性的值  
[[Put]](PropertyName Value)  
设置指定的属性值  
[[CanPut]](PropertyName)  
返回是否可以执行[[Put]]指定属性的操作的Boolean值  
[[HasProperty]](PropertyName)  
返回对象是否存在指定的属性名的Boolean值  
[[Delete]](PropertyName)  
移除对象指定的属性  
[[DefaultValue]](Hint)  
返回对象的默认值，只能是原始值，不允许对象和引用。  
[[Construct]] a list of argument values provied by the caller  
通过调用new操作符构造一个对象。  
[[Call]] a list of argument values provied by the caller  
通过函数调用表达式执行与对象关联的代码。  
[[HasInstance]](Value)  
返回一个Boolean值，表示This对象是否存在指定的委派（实例），仅对ECMAScript native中的Function对象实现。  
[[Scope]]   
作用域链，用于表示一个函数对象执行时的环境。  
[[Match]](String,Index)  
为正则表达式匹配和返回一个MatchResult的值。  
  
**ECMAScript Execution Context and Scope Chain**  
  
**Click here to open new window
CTRL+Mouse wheel to zoom in/out**  
  
Scope Chain DEMO:  
<script type="text/javascript">  
var v = 'global';  
  
var x = function(v){  
alert(v);  
with({}){  
    v = 'w';  
}  
try{alert(me);}catch(e){}  
};  
  
var y = function(){  
alert(v);  
};  
  
var z = function(){  
var v = 'z';  
y();  
alert(v);  
};  
  
var n = function(){  
var i = 0;  
var inner = function(){  
    alert(++i);  
};  
return inner;  
};  
  
x();  
  
z();  
  
var fn = n();  
  
fn();  
</script>  
  
  
**ECMA-262参考资料**  
ECMA-262 3rd Edition  
[http://www.ecma-international.or ... MA-ST-ARCH/ECMA-262](http://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262),%203rd%20edition,% 20December%201999.pdf  
  
ECMA-262 5th Edition  
[http://www.ecma-international.or ... CMA-ST/ECMA-262.pdf](http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf)  
  
What is Ecma International  
<http://www.ecma-international.org/memento/index.html>