JavaScript. The core

An object

ECMAScript是高度抽象的面向对象语言

A prototype chain

原型是一个有属性的对象，原型也具有自己的原型。如果一个对象的原型拥有一个非空的原型(non-null),那么这种**链式关系**被称作**原型链**。

原型链是对象有穷的链式结构，用于实现**继承**和**属性**共享。

注意，继承方法中的**"this"**将指向原始对象**(谁调用，指向谁)**，而不是方法被定义的类。

有时，一些对象具有相同的数据结构，但是拥有不同的值。这是我们就需要用构造函数按指定模式创建对象。

Constructor

function Foo(){}

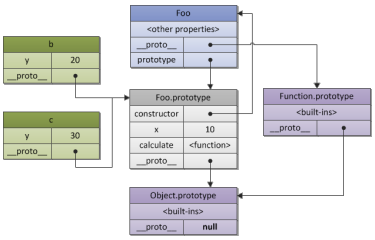
Foo 作为函数

Foo 作为构造函数

Foo.prototype 是一个对象

Foo实例指向 Foo.prototype对象

Foo.prototype对象的 __proto__属性指向 Object.prototype



Execution context stack

ECMAScript代码类型分为3类：**全局代码**，**方法代码**和**eval代码**。每种代码都在他

们自己的执行上下文中运行。**全局上下文只有一个**，而**方法实力和eval执行上下文可能有多个**。当调用方法时，会进入方法**执行上下文**执行方法中的代码。当调用eval时，会进入eval**执行上下文**，并执行eval中的代码。

caller 调用者

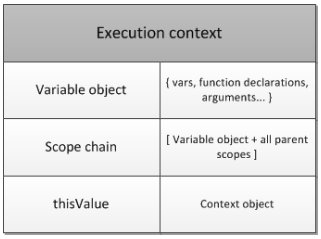
callee 被调用者

callee 在某些时候也能成为 callee的caller,例如：

栈中每个**执行上下文**都可以视为**对象**。

Execution context

执行上下文可被简单的看作为一个对象，每一个执行上下文的属性集合（或者称之为上下文状态）都将追踪其相关代码的运行状态。



除了以上三个必备属性（变量对象，this值，作用域链）,执行上下文可能会更具实现添加一些新状态(属性)

Variable object

变量对象是一系列与执行相关的数据。它是上下文一个比较特殊的对象，存储了上下文定义的变量和方法声明。

注意，**方法的表达式**并没有包含在变量对象中。
function bar() {} // 函数声明 FD
(function baz() {}); // 函数表达式 FE

那方法的变量对象是什么？在方法上下文中，变量对象被称为活动对象。

Activation object

当方法被调用时，一种被称为**活动对象**的特殊对象被创建。他是由一些普通的参数(方法内部变量和方法参数)和一个argumets对象(是具有index能力的参数集合，伪数组)构成。**活动对象**在方法中充当**变量对象** 的角色

也就是说，方法的变量对象和普通变量对象一样，但是除了变量和方法申明外，它

还存储了**形参**和**arguments**对象，同时被称为**活动对象**

就像原型链一样，我们可以将这一系列**变量对象**看为**作用域对象**，而它真正名称是**作用域链**

Scope chain

作用域链式是用于搜索上下文代码中标示符的对象列表

标示符是指：变量名称，方法定义(方法名)，形参等。

Closures

- 静态作用域链和动态作用域链的区别?
- 为什么会有闭包
-

ECMAScript中，方法是一级对象。

这说明**方法**可作为其他方法的参数（这种**方法**被称为**funargs**）。

接收funargs的方法被称为**高级方法**，按数学术语来说，称为operators。

同时方法也会作为其他方法的返回值。返回值为方法的方法被称为**方法值方法**。

第一种"funarg problem"被称为"upward funarg problem"（向上方法参数问题），这种情况通常是一个引用自由变量的方法被另一方法"向上"返回.

切忌-当方法创建的时刻-便保存了父级的作用域链，之后方法被调用时，这个被存储的作用域链将用于搜索变量。

"funarg problem"的第二种类型被称为"downward funarg problem"(向下方法参数问题)，在这类问题中，父级上下文已被弹出，这时在处理相关标示符的时候有些模糊。

由此得出结论，静态作用域是语言形成闭包的必备条件。

为什么有闭包?

This value