**[全面理解javascript的caller,callee,call,apply概念](http://www.cnblogs.com/jenry/archive/2008/02/14/1068840.html)**

在提到上述的概念之前，首先想说说javascript中函数的隐含参数：arguments

Arguments

该对象代表正在执行的函数和调用它的函数的参数。

[function.]arguments[n]  
参数function ：选项。当前正在执行的 Function 对象的名字。 n ：选项。要传递给 Function 对象的从0开始的参数值索引。   
说明

Arguments是进行函数调用时，除了指定的参数外，还另外创建的一个隐藏对象。Arguments是一个类似数组但不是数组的对象，说它类似数组是因为其具有数组一样的访问性质及方式，可以由arguments[n]来访问对应的单个参数的值，并拥有数组长度属性length。还有就是arguments对象存储的是实际传递给函数的参数，而不局限于函数声明所定义的参数列表，而且不能显式创建 arguments 对象。arguments 对象只有函数开始时才可用。下边例子详细说明了这些性质:

//arguments 对象的用法。  
function ArgTest(a, b){  
   var i, s = "The ArgTest function eXPected ";  
   var numargs = arguments.length;     // 获取被传递参数的数值。  
   var expargs = ArgTest.length;       // 获取期望参数的数值。  
   if (expargs < 2)  
      s += expargs + " argument. ";  
   else  
      s += expargs + " arguments. ";  
   if (numargs < 2)  
      s += numargs + " was passed.";  
   else  
      s += numargs + " were passed.";  
   s += "\n\n"  
   for (i =0 ; i < numargs; i++){      // 获取参数内容。  
   s += "  Arg " + i + " = " + arguments[i] + "\n";  
   }  
   return(s);                          // 返回参数列表。  
}

在此添加了一个说明arguments不是数组(Array类)的代码:

Array.prototype.selfvalue = 1;  
alert(new Array().selfvalue);  
function testAguments(){  
    alert(arguments.selfvalue);  
}

运行代码你会发现第一个alert显示1，这表示数组对象拥有selfvalue属性，值为1，而当你调用函数testAguments时，你会发现显示的是“undefined”，说明了不是arguments的属性，即arguments并不是一个数组对象。

 caller  
  返回一个对函数的引用，该函数调用了当前函数。  
  functionName.caller   
  functionName 对象是所执行函数的名称。  
说明  
对于函数来说，caller 属性只有在函数执行时才有定义。如果函数是由顶层调用的，那么 caller 包含的就是 null 。如果在字符串上下文中使用 caller 属性，那么结果和 functionName.toString 一样，也就是说，显示的是函数的反编译文本。  
下面的例子说明了 caller 属性的用法：

// caller demo {  
function callerDemo() {  
    if (callerDemo.caller) {  
        var a= callerDemo.caller.toString();  
        alert(a);  
    } else {  
        alert("this is a top function");  
    }  
}  
function handleCaller() {  
    callerDemo();  
}

callee

    返回正被执行的 Function 对象，也就是所指定的 Function 对象的正文。

[function.]arguments.callee  
可选项 function 参数是当前正在执行的 Function 对象的名称。

说明

callee 属性的初始值就是正被执行的 Function 对象。

callee 属性是 arguments 对象的一个成员，它表示对函数对象本身的引用，这有利于匿名  
函数的递归或者保证函数的封装性，例如下边示例的递归计算1到n的自然数之和。而该属性  
仅当相关函数正在执行时才可用。还有需要注意的是callee拥有length属性，这个属性有时候  
用于验证还是比较好的。arguments.length是实参长度，arguments.callee.length是  
形参长度，由此可以判断调用时形参长度是否和实参长度一致。

示例

//callee可以打印其本身  
function calleeDemo() {  
    alert(arguments.callee);  
}  
//用于验证参数  
function calleeLengthDemo(arg1, arg2) {  
    if (arguments.length==arguments.callee.length) {  
        window.alert("验证形参和实参长度正确！");  
        return;  
    } else {  
        alert("实参长度：" +arguments.length);  
        alert("形参长度： " +arguments.callee.length);  
    }  
}  
//递归计算  
var sum = function(n){  
  if (n <= 0)                          
  return 1;  
  else  
    return n ＋arguments.callee(n - 1)  
}  
比较一般的递归函数：

var sum = function(n){  
    if (1==n) return 1;  
else return n + sum (n-1);

调用时：alert(sum(100));  
其中函数内部包含了对sum自身的引用，函数名仅仅是一个变量名，在函数内部调用sum即相当于调用  
一个全局变量，不能很好的体现出是调用自身，这时使用callee会是一个比较好的方法。

apply and call

   它们的作用都是将函数绑定到另外一个对象上去运行，两者仅在定义参数方式有所区别：

    apply(thisArg,argArray);

    call(thisArg[,arg1,arg2…] ]);

即所有函数内部的this指针都会被赋值为thisArg，这可实现将函数作为另外一个对象的方法运行的目的

apply的说明

如果 argArray 不是一个有效的数组或者不是 arguments 对象，那么将导致一个 TypeError。  
如果没有提供 argArray 和 thisArg任何一个参数，那么 Global 对象将被用作 thisArg，   
并且无法被传递任何参数。

call的说明

call 方法可将一个函数的对象上下文从初始的上下文改变为由 thisArg指定的新对象。  
如果没有提供 thisArg参数，那么 Global 对象被用作 thisArg

相关技巧：

应用call和apply还有一个技巧在里面，就是用call和apply应用另一个函数（类）以后，当前的  
函数（类）就具备了另一个函数（类）的方法或者是属性，这也可以称之为“继承”。看下面示例:

// 继承的演示  
function base() {  
    this.member = " dnnsun\_Member";  
    this.method = function() {  
        window.alert(this.member);  
    }  
}  
function extend() {  
    base.call(this);  
    window.alert(member);  
    window.alert(this.method);  
}

上面的例子可以看出，通过call之后，extend可以继承到base的方法和属性。

顺便提一下，在Javascript框架prototype里就使用apply来创建一个定义类的模式，

其实现代码如下：

var Class = {  
  create: function() {  
    return function() {  
      this.initialize.apply(this, arguments);  
    }  
  }  
}  
解析：从代码看,该对象仅包含一个方法：Create，其返回一个函数，即类。但这也同时是类的  
构造函数，其中调用initialize，而这个方法是在类创建时定义的初始化函数。通过如此途径，  
就可以实现prototype中的类创建模式

示例：

var vehicle=Class.create();  
vehicle.prototype={  
    initialize:function(type){  
        this.type=type;  
    }  
    showSelf:function(){  
        alert("this vehicle is "+ this.type);  
    }  
}

var moto=new vehicle("Moto");  
moto.showSelf();

更详细的关于prototype信息请到其官方网站查看。