JavaScript 面向对象编程技术交流(提纲及参考资料)

主讲：EasyJF 开源团队(www.easyjf.com) 大峡

Javascript 是一种面向(基于)对象的动态脚本语言，是一种基于对象(Object)和事件驱动(EventDriven)并具有安全性能的脚本语言。他具有面向对象语言所特有的各种特性，比如封装、继承及多态等。但对于大多数人说，我们只把 javascript 做为一个函数式语言，只把他用于一些简单的前端数据输入验证以及实现一些简单的页面动态效果等，我们没能完全把握动态语言的各种特性。而 ext 中大量使用了 javascript 的面向对象特性，要使用好 ext 技术，javascript 的面向对象语言特性是我们必须完全把握的。

一、Javascript 中的数据类型

在了解 javascript 中的类及对象之前，我们先来看看 javascript 中的数据类型。有三种基本(原始)类型：数字、字符串以及布尔，另外还有 null(空)及 undefined(未定义)也可算是数据类型，而且每种只定义了单个值。

1、基本数据类型

数字(Numbers)

数字类型是所有语言中都存在的基本数据类型，javascript 中的数字类型主要包括整型(Int)与浮点型(Float)两种，但实质两种类型都是以浮点的形式保存在内存中。数字类型在 javascript 中一般与数字常量的形式出现在程序中，一般情况下是基于 10 进制的数据，由 0-9 这 10 个数字组成，比如 110、150 等，而也可以是以 0x 开头（由 0-9 以及 a 到 f 等 16 个字符组成）的 16 进制数据，比如 0xff 转换成 10 进制就是 255(即 15*16 + 15 = 255)；一些 javascript 实现还支持 8 进制数据，也就是以 0 开头的数据(由 0-7 这 8 个数字组成)，比如 0377 这个八进制数据转换成 10 进制就是 255，即（ 3*64 + 7*8 + 7 = 255 ）。

字符串(Strings)

字符串由各种字符、数字及特殊字符串组成，可以在程序中直接使用单引号或双引号来生成字符串常量。字符串中不能有回车符，要在字符串包含回车需要使用转义字符\n。下面是一些简单的字符串常量：

```
""    // The empty string: it has zero characters
'testing'
"3.14"
'name="myform"'
"Wouldn't you prefer O'Reilly's book?"
"This string\nhas two lines"
"π is the ratio of a circle's circumference to its diameter"
```

布尔 Boolean

布尔类型用来表示真或假，在 javascript 中，当用于布尔运算时，除了 0、空字符、null、undefined、NaN 等以外的数据都是表示真。

```
if(0||""||false||null||undefined||NaN)alert("有一个条件返回 true 了");
```

布尔常量只有 false 及 true，False 及 True 不是常量。

特殊值(null、undefined、NaN)

２、对象类型(Object)

对象属于复杂的数据类型，对象下面可以包含基本类型、对象、函数等，数组是一种对象类型。对于 javascript 来说，可以说一切都是对象，包括类！。

var c=new Object();

３、函数(Function)

函数是一种非常特殊的对象类型，函数中的内容是可执行代码，一个函数可发执行一系列的操作。与数组类似，作为特殊的数据类型函数可以有独特的语法表示格式。

正统的函数定义：

var f=function(){ };

改进后的定义：

function f(){};

匿名函数

var obj=function(){return 123}();

alert(obj);//输出：

函数参数的可变性

function add(s,b){

if(s)alert(“第一个参数是:”+s);

if(!b)alert(“没有第二个参数!”);

else alert(“第二个参数是:”+b);

}

arguments

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。

如何写一个方法，能实现任意多个数的求和?

alert(sum(1,2,3));//输出 6

alert(sum(100,200,500,900));//输出 1700

４、Prototype 属性

Prototype，原型，通过给出一个原型对象来指明所要创建的对象的类型，然后用这个原型对象的方法创建出更多同类型的对象，原始模型模式属于对象的创建模式。

在 javascript 中，prototype 是针对于某一类的对象的方法，它是一个给类的对象添加方法的方法！所有 JScript 内部对象都有只读的 prototype 属性。可以向其原型中动态添加功能(属性和方法)，但该对象不能被赋予不同的原型。然而，用户定义的对象可以被赋给新的原型。

简单的例子：

(1) Number.add(num)：作用，数字相加

实现方法：Number.prototype.add = function(num){return(this+num);}
试验：alert((3).add(15)) -> 显示 18
(2) Boolean.rev(): 作用，布尔变量取反
实现方法：Boolean.prototype.rev = function(){return(!this);}
试验：alert((true).rev()) -> 显示 false

二、类与对象

首先，我们来看看 javascript 语言中的类及对象。

具有相同或相似性质的对象的抽象就是类，可以理解为“类别”或者“类型”。因此，对象的抽象是类，类的具体化就是对象，也可以说类的实例是对象。类具有属性，它是对象的状态的抽象，用数据结构来描述类的属性；类具有操作，它是对象的行为的抽象，用操作名和实现该操作的方法来描述。比如“人”这种动物就是一个类，而具体某一个人就是“人”这个类的一个实例，“人”可以有许多实例（地球人超过六十亿了），但“人”这个类只有一个。

javascript 中类的定义：

```
function Animal(name){
this.name=name;
this.age=0;
    };
```

数组类似的语法定义类及对象

```
var c={
name:"旺财",
age:10,
run:function(){
    document.write(this.name+"在跑...");
    }
}
c.run();
```

c 是一个匿名类的唯一实例化对象，不能再进行实例化。

另外一种匿名类

```
var c=new function(){
this.name="旺财",
this.age=10,
this.run=function(){
    document.write(this.name+"在跑...");
    }
};
c.run();
```

javascript 中对象的创建

方法 1 ：像数组一样申明对象；

```
var myObject = {
username : "beansoft",
```

age : 24,
test : function() {alert(this.age);}
};
方法 2 ：直接用 new Object()来创建对象
var myObject = new Object();
myObject.username = "beansoft";
myObject.age = 24;
方法 3 ：通过类的构造函数来创建对象
function MyObject(username, age) {
this.username = username;
this.age = age;
this.test = function() {alert(this.age);};
}

三、封装、继承及多态

封装：封装，也就是把客观事物封装成抽象的类，并且类可以把自己的数据和方法只让可信的类或者对象操作，对不可信的进行信息隐藏。(写出人类 Person 的封装实现？)

继承：面向对象编程 (OOP) 语言的一个主要功能就是“继承”。继承是指这样一种能力：它可以使用现有类的所有功能，并在无需重新编写原来的类的情况下对这些功能进行扩展。(写一个简单的继承实现？)

方法重载(overloading)及覆盖(overriding)：Javascript 中的方法本身就是一个带可变参数的，不支持重载操作。但我们可以在方法体内自由检测方法的参数情况，来实现重载的效果。(使用可变参数或 arguments 来模拟重载的示例)。

覆盖，也称为改写，就是指子类中定义的方法替换掉父类的方法。(方法覆盖的示例？)

多态性(polymorphisn)：是允许你将父对象设置成为和一个或更多的他的子对象相等的技术，赋值之后，父对象就可以根据当前赋值给它的子对象的特性以不同的方式运作。简单的说，就是一句话：允许将子类类型的指针赋值给父类类型的指针。多态是为了实现另一个目的——接口重用!多态的作用，就是为了类在继承和派生的时候，保证使用“家谱”中任一类的实例的某一属性时的正确调用。

四、作用域 scope

先来看下面的 javascript：

```
var b1={v:"this is b1"};
var b2={v:"this is b2"};
function b()
{
   alert(this.v);
}
b();//输出：
window.b();//输出：
b.call(b1);//输出：
```

```
b.call(b2);//输出：
```

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

重点(the point)：

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。

在浏览执行的 javascript 中，默认情况下对象的作用域为 window。

```
c.run();
window.c.run();
```

五、动态语言的灵性

Javascript 作为一种动态语言，具有非常灵活的我发，在使用的过程中需要灵活掌握及应用他的动态特性，才会得心应手。

思考下面的输出

```
function Animal(name){
this.name=name;
this.age=0;
    };
var a1=Animal;//输出：
var a2=Animal();//输出：
var a3=new Animal();//输出：
var a4=new Animal;//输出：
```

本参考资料中各种问题的相关的答案及讨论，请到 www.vifir.com

六、其它参考资料

1、一个简单的 javascript 类定义例子

涵盖了 javascript 公有成员定义、私有成员定义、特权方法定义的简单示例！

```
<script>
    //定义一个 javascript 类
    function JsClass(privateParam/*    */,publicParam){//构造函数
        var priMember = privateParam;     //私有变量
        this.pubMember = publicParam;     //公共变量
        //定义私有方法
        function priMethod(){
            return "priMethod()";
        }
        //定义特权方法
        //特权方法可以访问所有成员
        this.privilegedMethod = function(){
            var str = "这是特权方法，我调用了\n";
            str += "        私有变量：" + priMember +"\n";
            str += "        私有方法：" + priMethod() +"\n";
            str += "        公共变量：" + this.pubMember +"\n";
            str += "        公共方法：" + this.pubMethod();

            return str;
        }
    }
    //添加公共方法
    //不能调用私有变量和方法
    JsClass.prototype.pubMethod = function(){
        return "pubMethod()";
    }

    //使用  JsClass 的实例
    JsObject = new JsClass("priMember","pubMember");

    //alert(JsObject.pubMember);//弹出 pubMember 信息
    //alert(JsObject.priMember);//弹出 undefined 信息
    //alert(JsObject.pubMethod());//弹出 pubMethod 信息
    //alert(JsObject.priMethod());//弹出"对象不支持此属性或方法"的错误
    alert(JsObject.privilegedMethod());
</script>
```

2、三种定义 Javascript 类的方法

1. 使用函数

这可能是最普通的方法之一。先定义一个函数，然后用其他关键字创建一个对象。

```
function App(type){
    this.type = type;
    this.color = "red";
    this.getInfo = getAppleInfo;
}
function getAppleInfo(){
    return this.color + ' ' + this.type + ' apple';
}
var apple = new Apple('macintosh');
apple.color = "reddish";
alert(apple.getInfo());
```

也可以直接在内部定义方法

```
function Apple(type){
    this.type = type;
    this.color = "red";
    this.getInfo = function(){
        return this.color + ' ' + this.type + ' apple';
    };
}
```

2. 使用 JSON

JSON 即 Javascript Object Notation。

```
var apple = {
    type: 'macintosh',
    color: 'red',
    getInfo: function(){
        return this.color + " + this.type + ' apple';
    }
};
```

用这种方法的话，你不需要也 不能创建类的实例，因为他已经存在了。直接拿来用就行了。

```
apple.color = 'reddish';
alert(apple.getInfo());
```

3. 使用函数创建单独的一个(匿名类)

此方法综合了上面的两个方法。你可以使用函数来定义单独的一个 class。

```
var apple = new function(){
    this.type = 'macintosh';
```

```
        this.color = 'red';
        this.getInfo = function(){
            return this.color + " + this.type + ' apple';
        };
    }
```

你会发现他和 1.1 十分类似，但是他使用对象的方法确和 2 很像。

```
    apple.color = 'reddish';
    alert(apple.getInfo());
```

3、JavaScript 定义类

在 javascript 中定义类或对象有下面几种常用的方式：

（1）工厂方式

```
function createCar(color, doors, mpg){
    var tempCar = new Object;
    tempCar.color = color;
    tempCar.doors = doors;
    tempCar.mpg = mpg;
    tempCar.showColor = function (){
        alert(this.color);
    };
    return tempCar;
}
var car1 = createCar("red", 4, 23);
var car2 = createCar("blue", 3, 25);
car1.showColor();
car2.showColor();
```

定义了一个能创建并返回特定类型对象的工厂函数，看起来还是不错的，但有个小问题，每次调用时都要创建新函数 showColor，我们可以把它移到函数外面，

```
function showColor(){
    alert(this.color);
}
```

在工厂函数中直接指向它

```
tempCar.showColor = showColor;
```

这样避免了重复创建函数的问题，但看起来不像对象的方法了。

（2）构造函数方式

```
function Car(sColor, iDoors, iMpg){
    this.color = sColor;
    this.doors = iDoors;
    this.mpg = iMpg;
    this.showColor = function (){
```

```
            alert(this.color);
        };
    }
    var car1 = new Car("red", 4, 23);
    var car2 = new Car("blue", 3, 25);
    car1.showColor();
    car2.showColor();
```

可以看到与第一中方式的差别，在构造函数内部无创建对象，而是使用 this 关键字。使用 new 调用构造函数时，先创建了一个对象，然后用 this 来访问。

这种用法于其他面向对象语言很相似了，但这种方式和上一种有同一个问题，就是重复创建函数。

（3）混合的构造函数/原型方式

这种方式就是同时使用构造函数方式和原型方式，综合他们的优点，构造函数方式前面介绍过了，现在看一下原型方式

```
function Car(){
}
Car.prototype.color = "red";
Car.prototype.doors = 4;
Car.prototype.mpg = 23;
Car.prototype.showColor = function(){
    alert(this.color);
};

var car1 = new Car();
var car2 = new Car();
```

首先定义了构造函数 Car，但无任何代码，然后通过 prototype 添加属性。优点：

a. 所有实例存放的都是指向 showColor 的指针，解决了重复创建函数的问题

b. 可以用 instanceof 检查对象类型

```
alert(car1 instanceof Car);   //true
```

缺点，添加下面的代码：

```
Car.prototype.drivers = new Array("mike", "sue");
car1.drivers.push("matt");

alert(car1.drivers);   //outputs "mike,sue,matt"
alert(car2.drivers);   //outputs "mike,sue,matt"
```

drivers 是指向 Array 对象的指针，Car 的两个实例都指向同一个数组。

下面就用一下混合方式：

```
function Car(sColor, iDoors, iMpg){
    this.color = sColor;
    this.doors = iDoors;
```

```
        this.mpg = iMpg;
        this.drivers = new Array("mike", "sue");
    }
    Car.prototype.showColor = function (){
        alert(this.color);
    };
    var car1 = new Car("red", 4, 23);
    var car2 = new Car("blue", 3, 25);

    car1.drivers.push("matt");
    alert(car1.drivers);
    alert(car2.drivers);
```

这样就没有问题了，并且还可以使用 instanceof

（4）动态原型法

```
    function Car(sColor, iDoors, iMpg){
        this.color = sColor;
        this.doors = iDoors;
        this.mpg = iMpg;
        this.drivers = new Array("mike", "sue");
        if(typeof Car.initialized == "undefined"){
            Car.prototype.showColor = function (){
                alert(this.color);
            };
            Car.initialized = true;
        }
    }
    var car1 = new Car("red", 4, 23);
    var car2 = new Car("blue", 3, 25);

    car1.drivers.push("matt");
    alert(car1.drivers);
    alert(car2.drivers);
```

这种方式是我最喜欢的，所有的类定义都在一个函数中完成，看起来非常像其他语言的类定义，不会重复创建函数，还可以用 instanceof

4、javascript 继承实现方法

javascript 的继承机制并不是明确规定的,而是通过模仿实现的,意味着继承不是由解释程序处理,开发者有权决定最适合的继承方式. 下面我给出几种常用的方法:

1.对象冒充

原理: 构造函数使用 this 关键字给所有属性和方法赋值, 因为构造函数只是一个函数,所以可以使 ClassA 的构造函数成为 classB 的方法,然后调用它.这样 classB 就会收到 classA 的

构造函数中定义的属性和方法.例子:

```
function classA(name) {
        this.name=name;
        this.showName=function(){alert(this.name);}
}
function classB(name) {
        this.newMethod = classA;
        this.newMethod(name);
}
  obj = new classA("hero");
  objB = new classB("dby");
  obj.showName(); //    print hero
  objB.showName(); // print dby   说明 classB 继承了 classA 的方法.
  对象冒充可以实现多重继承   例如
  function classz(){
  this.newMethod = classX;
  this.newMethod();
  delete this.newMethod;
  this.newMethod=classY;
  this.newMethod():
  delete this.newMethod;
 }
```

但是如果 classX 和 classY 有相同的属性或者方法,classY 具有高优先级.

2.call()方法

call 方法使与经典的对象冒充法就相近的方法,它的第一个参数用作 this 的对象,其他参数都直接传递给函数自身.

```
  function sayName(perfix) {
  alert(perfix+this.name);
   }
   obj= new Object();
   obj.name="hero";
   sayName.call(obj,"hello," );
   function classA(name) {
        this.name=name;
        this.showName=function(){alert(this.name);};
   }
   function classB(name) {
  classA.call(this,name);
   }
   objB = new classB("bing");
   objB.showName();////说明 classB 继承 classA 的 showName 方法
```

3.apply()方法

aplly()方法有 2 个参数,一个用作 this 对象,一个使传递给函数的参数数组.

```
function sayName(perfix)  {
alert(perfix+this.name);
 }
 obj= new Object();
 obj.name="hero";
 sayName.aplly(obj,new Array("hello,") );
```

4. 原型链

prototype 对象的任何属性和方法都会被传递给对应类的所有实例,原型链就是用这种方式来显现继承.

```
function classA (){}
classA.prototype.name="hero";
classA.prototype.showName=function(){alert(this.name)}
function classB(){}
classB.prototype=new classA();
objb = new classB()
objb.showName();//print hero   说明 b 继承了 a 的方法
```

这里需要注意 调用 classA 的构造函数时,没有给它传递参数,这是原型链的标准做法,确保函数的构造函数没有任何参数.

并且 子类的所有属性和方法,必须出现在 prototype 属性被赋值后,应为在它之前赋的值会被删除.因为对象的 prototype 属性被替换成了新对象,添加了新方法的原始对象将被销毁.

5 混和方式

就是用冒充方式 定义构造函数属性,用原型法定义对象方法.

```
function classA(name) {
   this.name=name;
}
classA.prototype.showName=function(){alert(this.name)}
function classB(name) {
   classA.call(this,name);
}
classB.prototype = new classA();
classB.prototype.showName1=function(){alert(this.name+"*****");};
obj = new classB("hero");
obj.showName();
obj.showName1();
```

在 classB 的构造函数中通过调用 call 方法 继承 classA 中的 name 属性,用原型链来继承 classA 的 showName 方法.

5、面向对象概念摘要

角度 1

封装的概念好比一辆汽车，你学开车的时候只需学会诸如踩油门、刹车，转方向盘即可，无需去了解它的发动机是如何发动。

继 承，先说说我对类的理解，类起到的作用有：分类（你所用某个类创建的对象实际上该类的个案）和模板的作用，那么继承则起到了对类再次分类的作用，比如，有 个类“动物”，“哺乳动物”继承“动物”，再往下“马”又继承了“哺乳动物”这个类。在这里，我们从下往上讲，首先，我们把某种东西划分出来，叫做“马” （当然还有“牛”，“鱼”等等），接着，我们发现，“马”，“羊”等还有很多共同的特点，于是，我们再次划分出了“哺乳动物”这个类，再次分类，我们则有 了“动物”。但在实际开发中，我们一般是从上往下定义的，即先有了“动物”，再有“哺乳动物”，最后有“马”。

多态，正如上面朋友所说一重写，二重载。用汉字来打个比方，比如“开”这个字，在不同的时候各有的意思，比如“开门”，“开窗”，甚至有“开车”，“开饭”等，具有相同名称但操作不同。具体的实现我就不累述了。

说说接口吧，在 JAVA 不支持多继承的，实际上接口起到了类似多继承的作用，一个类只能继承另一个类（或抽象类）但可以实现多个接口。打个比方，“张三”， 他是一个“人”，因此他继承了“人”；与此同时，他是一个“司机”，他的平时行为还有“开车”，很显然，这并不能从“人”这个类里继承下来。怎么办？ JAVA 里只支持单继承，这个时候，接口就起到了作用，它定义了“司机”这个接口，“张三”实现了它，因此，他会开车了。

角度 2：

封装：就象一个盒子，你不需要知道里面有什么东西，只知道它有那些用处就行

继承：就象父子关系，儿子是父亲的继承

多态：好比一个父亲有多个儿子，各个儿子有不同的特征

角度 3：

封装(Encapsulation):封装是一个面向对象的概念,对外部世界,隐藏类的内部.封装优点:

1.好的封装能减少耦合.

2.类的内部的实现可以自由改变.

3.一个类有更清楚的接口.

Data Hiding(数据隐藏):封装的一个最有用的形式是数据隐藏.一个类的数据表现一个对象的状态.修饰符支持封装:

Private:只有类本身能存取.

Protected:类和派生类可以存取.

Internal:只有同一个项目中的类可以存取.

Protected Internal:是 Protected 和 Internal 的结合.

Public:完全存取.

other Encapsulating Strategy:(其他封装策略)属性和索引器的目的是封装一个类的细节和给类的用户提供一个公共的接口。封装和继承的关系:

封装的意思是包容(聚合),类与类之间的关系是"has a".一个类里面有另一个类.

继承,类与类之间的关系是"is a".

多态(Polymorphism):就是怎样重载一个虚拟类.多态是面向对象的重要概念.

5、面向对象方法概述

1、面向对象方法（Object Oriented）产生的原因

以前的开发方法，只是单纯地反映管理功能的结构状况，或者只是侧重反映事物的信息特征和信息流程，只能被动应和实际问题需要的做法。面向对象的方法把数据和过程包装成为对象，以对象为基础对系统进行分析与设计，为认识事物提供了一种全新的思路和办法，是一种综合性的开发方法。

2、基本思想

客观世界是由各种各样的对象组成的，每种对象都有各自的内部状态和运动规律，不同对象之间的相互作用和联系就构成了各种不同的系统。

对象(Object)是客观世界中的任何事物或人们头脑中的各种概念在计算机程序世界里的抽象表示。是面向对象程序设计的基本元素。

3、基本概念

客观世界由各种“对象”(Object)组成，任何客观事物都是对象，对象是在原事物基础上抽象的结果。任何复杂的事物都可以通过对象的某种组合结构构成。对象可由相对比较简单的对象以某种方式组成；

对象由属性和方法组成。属性(Attribute)反映了对象的信息特征,如特点、值、状态等等.而方法(Method)则是用来定义改变属性状态的各种操作；

对象之间的联系主要是通过传递消息(Message)来实现的，而传递的方式是通过消息模式(Message pattern)和方法所定义的操作过程来完成的；

对象可按其属性进行归类(Class)。类有一定的结构，类上可以由超类(Superclass)，类下可以有子类(Subclass)。对象或类之间的层次结构是靠继承关系(Inheritance)维系的；

对象是一个被严格模块化了的实体，称之为封装(Encapsulation)。这种封装了的对象满足软件工程的一切要求，而且可以直接被面向对象的程序设计语言所接受。

4、开发过程

系统调查和需求分析：对系统将要面临的具体管理问题以及用户对系统开发的需求进行调查研究，即先弄清要干什么的问题。

分析问题的性质和求解问题：在繁杂的问题域中抽象地识别出对象以及其行为、结构、属性、方法等。一般称之为面向对象的分析，即 OOA。

整理问题：对分析的结果作进一步的抽象、归类、整理，并最终以范式的形式将他们确定下来。一般称之为面向对象的设计，即 OOD。

程序实现：用面向对象的程序设计语言将上一步整理的范式直接映射（即直接用程序设计语言来取代）为应用软件。一般称之为面向对象的程序，即 OOP。

面向对象的方法：面向对象方法都支持三种基本的活动：识别对象和类，描述对象和类之间的关系，以及通过描述每个类的功能定义对象的行为。