# js自定义对象讲解

[java面试](http://blog.163.com/ainiyiwannian2046@126/blog/#m=0&t=1&c=fks_087071081086087074086094084095080095080067085085082067) 2009-04-22 16:57:07 阅读3 评论0 字号：大中小

## 概述

在Java语言中，我们可以定义自己的类，并根据这些类创建对象来使用，在Javascript中，我们也可以定义自己的类，例如定义User类、Hashtable类等等。

目前在Javascript中，已经存在一些标准的类，例如Date、Array、RegExp、String、Math、Number等等，这为我们编程提供了许多方便。但对于复杂的客户端程序而言，这些还远远不够。

与Java不同，Java2提供给我们的标准类很多，基本上满足了我们的编程需求，但是Javascript提供的标准类很少，许多编程需求需要我们自己去实现，例如Javascript没有哈西表Hashtable，这样的话在处理键值方面就很不方便。

因此，我个人认为一个完整的Javascript对象视图应该如下：

## 基本概念

### 自定义对象。

根据JS的对象扩展机制，用户可以自定义JS对象，这与Java语言有类似的地方。

与自定义对象相对应的是JS标准对象，例如Date、Array、Math等等。

### 原型（prototype）

在JS中，这是一种创建对象属性和方法的方式，通过prototype可以为对象添加新的属性和方法。

通过prototype我们可以为JS标准对象添加新的属性和方法，例如对于String对象，我们可以为其添加一个新的方法trim（）。

与严格的编程语言（例如Java）不同，我们可以在运行期间为JS对象动态添加新的属性。

## 语法规则

### 对象创建方式

#### 对象初始化器方式

格式：objectName = {property1:value1, property2:value2,…, propertyN:valueN}

property是对象的属性

value则是对象的值，值可以是字符串、数字或对象三者之一

例如：

var user={name:“user1”,age:18}；

 var user={name:“user1”,job:{salary:3000,title:programmer}

以这种方式也可以初始化对象的方法，例如：

    var user={name:“user1”,age:18,getName:function(){

                return this.name;

          }

    }

后面将以构造函数方式为重点进行讲解，包括属性和方法的定义等等，也针对构造函数的方式进行讲解。

#### 构造函数方式

编写一个构造函数，并通过new方式来创建对象，构造函数本可以带有构造参数

例如：

function User(name,age){

          this.name=name;

          this.age=age;

          this.canFly=false;

    }

    var use=new User();

### 定义对象属性

JS中可以为对象定义三种类型的属性：私有属性、实例属性和类属性，与Java类似，私有属性只能在对象内部使用，实例属性必须通过对象的实例进行引用，而类属性可以直接通过类名进行引用。

#### 私有属性定义

私有属性只能在构造函数内部定义与使用。

语法格式：var propertyName=value；

例如：

function User(age){

           this.age=age;

           var isChild=age<12;

           this.isLittleChild=isChild;

    }

    var user=new User(15);

    alert(user.isLittleChild);//正确的方式

    alert(user.isChild);//报错：对象不支持此属性或方法

#### 实例属性定义

实例属性定义也存在两种方式：

prototype方式，语法格式：functionName.prototype.propertyName=value

this方式，语法格式：this.propertyName=value，注意后面例子中this使用的位置

上面中value可以是字符创、数字和对象。

例如：

function User(){ }

User.prototype.name=“user1”;

User.prototype.age=18;

var user=new User();

alert(user.age);

—————————————–

function User(name,age,job){

         this.name=“user1”;

         this.age=18;

         this.job=job;

    }

    alert(user.age);

#### 类属性定义

语法格式：functionName.propertyName=value

例如：

function User(){ }

User.MAX\_AGE=200;

User.MIN\_AGE=0;

alert(User.MAX\_AGE);

参考JS标准对象的类属性：

Number.MAX\_VALUE //最大数值 Math.PI //圆周率

对于属性的定义，除了上面较为正规的方式外，还有一种非常特别的定义方式，语法格式： obj[index]=value

例子：

     function User(name){

             this.name=name;

             this.age=18;

             this[1]=“ok”;

             this[200]=“year”;

     }

     var user=new User(“user1”);

     alert(user[1]);

     在上面例子中，要注意：不同通过this[1]来获取age属性，也不能通过this[0]来获取name属性，即通过index方式定义的必须使用index方式来引用，而没有通过index方式定义的，必须以正常方式引用

### 定义对象方法

JS中可以为对象定义三种类型的方法：私有方法、实例方法和类方法，与Java类似：

私有方法只能在对象内部使用

实例方法必须在对象实例化后才能使用

类方法可以直接通过类名去使用

注意：方法的定义不能通过前面所说的index方式进行。

#### 定义私有方法

私有方法必须在构造函数体内定义，而且只能在构造函数体内使用。

语法格式：function methodName(arg1,…,argN){ }

例如：

function User(name){

          this.name=name;

          function getNameLength(nameStr){

              return nameStr.length;

          }

          this.nameLength=getNameLength(this.name);

    }

#### 定义实例方法

目前也可以使用两种方式：

prototype方式，在构造函数外使用，语法格式：

functionName.prototype.methodName=method;

或者

functionName.prototype.methodName=function(arg1,…,argN){};

this方式，在构造函数内部使用，语法格式：

this.methodName=method;

或者

this.methodName=function(arg1,…,argN){};

上面的语法描述中，method是外部已经存在的一个方法，methodName要定义的对象的方法，意思就是将外部的一个方法直接赋给对象的某个方法。

以function(arg1,…,argN){}的方式定义对象方法是开发人员应该掌握的。

定义实例方法的一些例子：例子1

function User(name){

       this.name=name;

       this.getName=getUserName;

       this.setName=setUserName;

}

function getUserName(){

       return this.name;

}

Function setUserName(name){

       this.name=name;

}

定义实例方法的一些例子：例子2

function User(name){

       this.name=name;

       this.getName=function(){

               return this.name;

       };

       this.setName=function(newName){

               this.name=newName;

       };

}

定义实例方法的一些例子：例子3

function User(name){

       this.name=name;

}

User.prototype.getName=getUserName;

User.prototype.setName=setUserName();

function getUserName(){

       return this.name;

}

Function setUserName(name){

       this.name=name;

}

定义实例方法的一些例子：例子4

function User(name){

       this.name=name;

}

User.prototype.getName=function(){

       return this.name;

};

User.prototype.setName=function(newName){

       this.name=newName;

};

#### 定义类方法

类方法需要在构造函数外面定义，可以直接通过构造函数名对其进行引用。

语法格式：

functionName.methodName=method;

或者

functionName.methodName=function(arg1,…,argN){};

例子：

function User(name){

            this.name=name;

    }

    User.getMaxAge=getUserMaxAge;

    function getUserMaxAge(){

           return 200;

    }

    或者

    User.getMaxAge=function(){return 200;};

    alert(User.getMaxAge());

### 属性与方法的引用

#### 从可见性上说

私有属性与方法，只能在对象内部引用。

实例属性与方法，可以在任何地方使用，但必须通过对象来引用。

类属性与方法，可以在任何地方使用，但不能通过对象的实例来引用（这与Java不同，在Java中静态成员可以通过实例来访问）。

#### 从对象层次上说

与Java bean的引用类似，可以进行深层次的引用。

几种方式：

简单属性：obj.propertyName

对象属性：obj.innerObj.propertyName

索引属性：obj.propertyName[index]

对于更深层次的引用与上面类似。

#### 从定义方式上说

通过index方式定义的属性，必须通过index方式才能引用。

通过非index方式定义的属性，必须通过正常的方式才能引用。

另外注意：对象的方法不能通过index方式来定义。

### 属性与方法的动态增加和删除

1）对于已经实例化的对象，我们可以动态增加和删除它的属性与方法，语法如下（假定对象实例为obj）：

动态增加对象属性

obj.newPropertyName=value;

动态增加对象方法

    obj.newMethodName=method或者=function(arg1,…,argN){}

动态删除对象属性

    delete obj.propertyName

动态删除对象方法

    delete obj.methodName

2）例子：

    function User(name){

             this.name=name;

             this.age=18;

    }

    var user=new User(“user1”);

    user.sister=“susan”;

    alert(user.sister);//运行通过

    delete user.sister;

    alert(user.sister);//报错：对象不支持该属性

    user.getMotherName=function(){return “mary”;}

    alert(user.getMotherName());//运行通过

    delete user.getMotherName;

    alert(user.getMotherName());//报错：对象不支持该方法

## 四，总结

1，自定义对象机制，是JS最为吸引人的机制之一，对于C++和Java程序员而言，这简直太棒了！

2，对象创建存在两种方式：对象初始化器和构造函数。

3，对象属性和方法，具有可见性的约束，不同可见性的属性和方法，其定义方式也不一样。

# js中this的用法

[javascript开发](http://blog.163.com/zhengjianweilx@126/blog/#m=0&t=1&c=fks_087067087085082065083094083095092087086066081083087071) 2008-02-04 09:47:11 阅读0 评论0 字号：大中小

       this指针是面向对象程序设计中的一项重要概念，它表示当前运行的对象。在实现对象的方法时，可以使用this指针来获得该对象自身的引用。

## javascript 代码中this 的用法

和其他面向对象的语言不同，JavaScript中的this指针是一个动态的变量，一个方法内的this指针并不是始终指向定义该方法的对象的，在上一节讲函数的apply和call方法时已经有过这样的例子。为了方便理解，再来看下面的例子：

＜script language="JavaScript" type="text/javascript"＞

＜!--

//创建两个空对象

var obj1=new Object();

var obj2=new Object();

//给两个对象都添加属性p，并分别等于1和2

obj1.p=1;

obj2.p=2;

//给obj1添加方法，用于显示p的值

obj1.getP=function(){

       alert(this.p); //表面上this指针指向的是obj1

}

//调用obj1的getP方法

obj1.getP();

//使obj2的getP方法等于obj1的getP方法

obj2.getP=obj1.getP;

//调用obj2的getP方法

obj2.getP();

//--＞

＜/script＞

从代码的执行结果看，分别弹出对话框显示1和2。由此可见，getP函数仅定义了一次，在不同的场合运行，显示了不同的运行结果，这是有this指针的变化所决定的。在obj1的getP方法中，this就指向了obj1对象，而在obj2的getP方法中，this就指向了obj2对象，并通过this指针引用到了两个对象都具有的属性p。

　　由此可见，JavaScript中的this指针是一个动态变化的变量，它表明了当前运行该函数的对象。由this指针的性质，也可以更好的理解JavaScript中对象的本质：一个对象就是由一个或多个属性（方法）组成的集合。每个集合元素不是仅能属于一个集合，而是可以动态的属于多个集合。这样，一个方法（集合元素）由谁调用，this指针就指向谁。实际上，前面介绍的apply方法和call方法都是通过强制改变this指针的值来实现的，使this指针指向参数所指定的对象，从而达到将一个对象的方法作为另一个对象的方法运行。

　　每个对象集合的元素（即属性或方法）也是一个独立的部分，全局函数和作为一个对象方法定义的函数之间没有任何区别，因为可以把全局函数和变量看作为window对象的方法和属性。也可以使用new操作符来操作一个对象的方法来返回一个对象，这样一个对象的方法也就可以定义为类的形式，其中的this指针则会指向新创建的对象。在后面可以看到，这时对象名可以起到一个命名空间的作用，这是使用JavaScript进行面向对象程序设计的一个技巧。例如：

代码片段

var namespace1=new Object();

namespace1.class1=function(){

      //初始化对象的代码

}

var obj1=new namespace1.class1();

　　这里就可以把namespace1看成一个命名空间。

　　由于对象属性（方法）的动态变化特性，一个对象的两个属性（方法）之间的互相引用，必须要通过this指针，而其他语言中，this关键字是可以省略的。如上面的例子中：

obj1.getP=function(){

       alert(this.p); //表面上this指针指向的是obj1

}

　　这里的this关键字是不可省略的，即不能写成alert(p)的形式。这将使得getP函数去引用上下文环境中的p变量，而不是obj1的属性。

Js this 的用法

this是JavaScript中功能最强大的关键字之一。不幸的是，如果你不知道它具体怎么工作，你将很难正确使用它。

## 网页事件处理中this 的用法

    下面我来阐述如何在[事件处理](http://www.quirksmode.org/js/introevents.html" \t "_blank)中来使用this，之后我会附加一些this相关的例子。

    Owner

    接下来文章中我们将要讨论的问题是：在函数doSomething()中this所指的是什么？

Java代码 [复制代码](http://www.javaeye.com/topic/24457)

function doSomething() {

  this.style.color = '#cc0000';

}

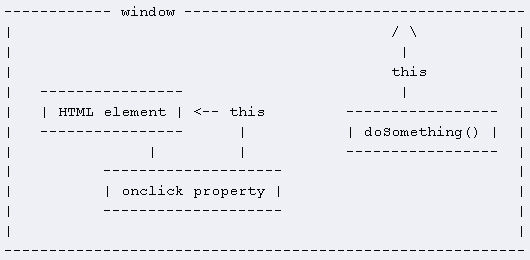
function doSomething() {

this.style.color = '#cc0000';

}

    在JavaScript中，this通常指向的是我们正在执行的函数本身（译者注：用owner代表this所指向的内容），或者是，指向该函数所属的对象。当我们在页面中定义了函数doSomething()的时候，它的owner是页面，或者是JavaScript中的window对象（或 global对象）。对于一个onclick属性，它为它所属的HTML元素所拥有，this应该指向该HTML元素。

    这种“所有权”就是JavaScript中面向对象的一种方式。在[Objects as associative arrays](http://www.quirksmode.org/js/associative.html" \t "_blank)中可以查看一些更多的信息。



    如果我们在没有任何更多准备情况下执行doSomething()，this关键字会指向window，并且该函数试图改变window的 style.color。因为window并没有style对象，这个函数将非常不幸的运行失败，并产生JavaScript错误。

    Copying

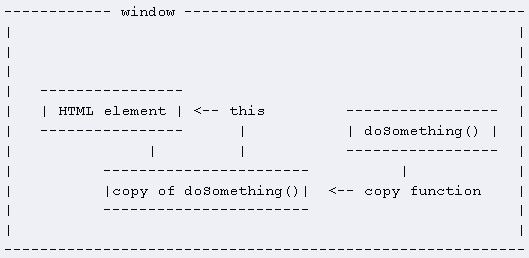
    因此如果我们想充分利用this，我们不得不注意使用this的函数应该被正确的HTML元素所拥有。换句话说，我们应该复制这个函数到我们的onclick属性。[Traditional event registration](http://www.quirksmode.org/js/events_tradmod.html" \t "_blank)会关心它。

Java代码 [复制代码](http://www.javaeye.com/topic/24457)

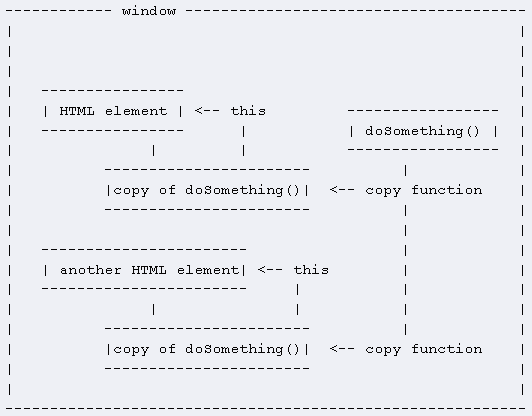
element.onclick = doSomething;

element.onclick = doSomething;

    这个函数被完整复制到onclick属性（现在成为了函数）。因此如果这个event handler被执行，this将指向HTML元素，并且该元素的颜色得以改变。



    这种方法使得我们能够复制这个函数到多个event handler。每次this都将指向正确的HTML元素：



    这样你就可以最大限度使用this。每当执行该函数，this所指向的HTML元素都正确响应事件，这些HTML元素拥有doSomething()的一个拷贝。

    Referring

    然而，如果你使用[inline event registration](http://www.quirksmode.org/js/events_early.html)(内联事件注册)

Java代码 [复制代码](http://www.javaeye.com/topic/24457)

<element onclick="doSomething()">

<element onclick="doSomething()">

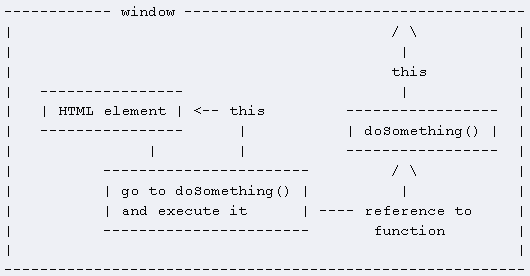
    你将不能拷贝该函数！反而这种差异是非常关键的。onclick属性并不包含实际的函数，仅仅是一个函数调用。

Java代码 [复制代码](http://www.javaeye.com/topic/24457)

doSomething();

doSomething();

    因此，它将声明“转到doSomething()并且执行它”。当我们到达doSomething()，this关键字又重新指向了全局的window对象，函数返回错误信息。



    The difference

    如果你想使用this来指向HTML元素响应的事件，你必须确保this关键字被写在onclick属性里。只有在这种情况下它才指向event handler所注册的HTML元素。

Java代码 [复制代码](http://www.javaeye.com/topic/24457)

element.onclick = doSomething;

alert(element.onclick)

element.onclick = doSomething;

alert(element.onclick)

    你将得到

Java代码 [复制代码](http://www.javaeye.com/topic/24457)

function doSomething() {

  this.style.color = '#cc0000';

}

function doSomething() {

this.style.color = '#cc0000';

}

    正如你所见，this关键字被展现在onclick函数中，因此它指向HTML元素。

    但是如果执行

Java代码 [复制代码](http://www.javaeye.com/topic/24457)

<element onclick="doSomething()">

alert(element.onclick)

<element onclick="doSomething()">

alert(element.onclick)

    你将得到

Java代码 [复制代码](http://www.javaeye.com/topic/24457)

function onclick() {

  doSomething()

}

function onclick() {

doSomething()

}

    这仅仅是到doSomething()函数的一个引用。this关键字并没有出现在onclick函数中，因此它不指向HTML元素。

    例子--拷贝

    下面的例子中，this被写入onclick函数里：

Java代码 [复制代码](http://www.javaeye.com/topic/24457)

element.onclick = doSomething

element.addEventListener('click', doSomething, false)

element.onclick = function() {this.style.color = '#cc0000';}

<element onclick="this.sytle.color = '#cc0000';">

element.onclick = doSomething

element.addEventListener('click', doSomething, false)

element.onclick = function() {this.style.color = '#cc0000';}

<element onclick="this.sytle.color = '#cc0000';">

    例子--引用

    下述情况中，this指向window：

Java代码 [复制代码](http://www.javaeye.com/topic/24457)

element.onclick = function() {doSomething()}

element.attachEvent('onclick', doSomething)

<element onclick="doSomething()">

element.onclick = function() {doSomething()}

element.attachEvent('onclick', doSomething)

<element onclick="doSomething()">

    注意attachEvent()的出现。[Microsoft event registration model](http://www.quirksmode.org/js/events_advanced.html" \t "_blank)最主要的弊端是attachEvent()创建了一个指向函数的引用，而不是复制它。因此有时不可能知道哪个HTML正在处理该事件。

    组合使用

    当使用内联事件注册时，你可以将this发送到函数以至于可以正常使用：

Java代码 [复制代码](http://www.javaeye.com/topic/24457)

<element onclick="doSomething(this)">

function doSomething(obj) {

  //this出现在event handler中并被发送到函数

  //obj指向HTML元素，因此可以这样：

  obj.style.color = '#cc0000';

}

<element onclick="doSomething(this)">

function doSomething(obj) {

//this出现在event handler中并被发送到函数

//obj指向HTML元素，因此可以这样：

obj.style.color = '#cc0000';

}