JavaScript 中函数与对象的解析

薛晓冬

摘　要： 在 Ajax 应用中，客户端的 JavaScript 编码越来越庞大，需要程序员有效地组织代码实现应用功能。JavaScript 语言与 C++、Java 等语言在语法上类似，但其函数具有独特性，深入理解 JavaScript 语言中的函数与对象是掌握 JavaScript 的基础。

关键词： JavaScript；函数；对象

1　引言

随着富客户端技术的发展，尤其是 Ajax 技术的广泛应用，客户端 JavaScript（后简写为 JS）编码越来越庞大。函数是进行模块化程序设计的基础，要编写复杂的 Ajax 应用程序，必须对函数有深入地了解。JS 中的函数不同于其他的语言，它的函数都是作为一个对象被维护和运行的。通过函数对象的性质，可以很方便地将一个函数赋值给一个变量或者将函数作为参数传递。

先看一下函数的使用语法：

```
function func1(…){…}
var func2=function(…){…};
var func3=function func4(…){…};
var func5=new Function();
```

这些都是 JS 声明函数的正确语法，它们和其他语言中常见的函数定义方式有着很大的区别。

2　函数对象（Function Object）

可以用 function 关键字定义一个函数，并为每个函数指定一个函数名，通过函数名来进行调用。在 JS 解释执行时，函数都是被维护为一个对象，这就是要介绍的函数对象。

函数对象与其他用户所定义的对象有着本质的区别，这一类对象被称之为内部对象，例如日期对象（Date）、数组对象（Array）、字符串对象（String）都属于内部对象。这些内置对象的构造器是由 JS 本身所定义的，通过执行 new Array () 这样的语句返回一个对象，JS 内部有一套机制来初始化返回的对象，而不是由用户来指定对象的构造方法。

在 JS 中，函数对象对应的类型是 Function，正如数组对象对应的类型是 Array，日期对象对应的类型是 Date 一样，可以通过 new Function () 来创建一个函数对象，也可以通过 function 关键字来创建一个对象。为了便于理解，比较函数对象的创建和数组对象的创建。先看数组对象，下面两行代码都是创建一个数组对象 myArray：

```
var myArray=[];
//等价于
var myArray=new Array();
//同样，下面的两段代码也都是创建一个函数 myFunction：
function myFunction(a,b){
    return a+b;
}
//等价于
var myFunction=new Function("a","b","return a+b");
```

通过和构造数组对象语句的比较，可以清楚地看到 JS 函数作为对象的本质，在解释器内部，当遇到这种语法时，就会自动构造一个 Function 对象，将函数作为一个内部的对象来存储和运行。一个函数对象名称（函数变量）和一个普通变量名称具有同样的规范，都可以通过变量名来引用这个变量，但是函数变量名后面可以加括号和参数列表来进行函数调用。

用 new Function () 的形式来创建一个函数不常见，因为一个函数体通常会有多条语句，如果将它们以一个字符串的形式作为参数传递，代码的可读性差。下面介绍一下其使用语法：

```
var funcName=new Function(p1,p2,...,pn,body);
```

参数的类型都是字符串，p1 到 pn 表示所创建函数的参数名称列表，body 表示所创建函数的函数体语句，funcName 就是所创建函数的名称。可以不指定任何参数创建一个空函数，不指定 funcName 创建一个无名函数，当然那样的函数没有任何意义。

需要注意的是，p1 到 pn 是参数名称的列表，即 p1 不仅能代表一个参数，它也可以是一个逗号隔开的参数列表，例如下面的定义是等价的：

```
new Function("a", "b", "c", "return a+b+c")
new Function("a, b, c", "return a+b+c")
new Function("a,b", "c", "return a+b+c")
```

JS 引入 Function 类型并提供 new Function () 这样的语法是因为函数对象添加属性和方法就必须借助于 Function 这个类型。

JS 的函数本质是一个内部对象，由 JS 解释器决定其运行

方式。通过上述代码创建的函数，在程序中可以使用函数名进行调用，文中开头列出的函数定义问题也得到了解释。注意可直接在函数声明后面加上括号表示创建完成后立即进行函数调用，例如：

```
var i=function (a,b){
    return a+b;
}(1,2);
alert(i);
```

这段代码会显示变量 i 的值等于 3。i 是表示返回的值，而不是创建的函数，因为括号“(”比等号“=”有更高的优先级。这样的代码可能并不常用，但当用户想在很长的代码段中进行模块化设计或者想避免命名冲突，这是一个不错的解决办法。

需要注意的是，尽管下面两种创建函数的方法是等价的：

```
function funcName(){
  //函数体
}
//等价于
var funcName=function(){
    //函数体
}
```

但前面一种方式创建的是有名函数，而后面是创建了一个无名函数，只是让一个变量指向了这个无名函数。在使用上仅有一点区别，就是对于有名函数，它可以出现在调用之后再定义；而对于无名函数，它必须是在调用之前就已经定义。

```
<script language="JavaScript" type="text/javascript">
    //<! --
func();
var func=function(){
    alert(1)
}
//--></script>
```

这段语句将产生 func 未定义的错误。

```
<script language="JavaScript" type="text/javascript">
    //<! --
func();
function func(){
    alert(1)
}
//--></script>
```

下面的语句则能正确执行：

```
<script language="JavaScript" type="text/javascript">
    //<! --
func();
var someFunc=function func(){
    alert(1)
}
//--></script>
```

由此可见，尽管 JS 是一门解释型的语言，但它会在函数调用时，检查整个代码中是否存在相应的函数定义，这个函数名只有是通过 function funcName () 形式定义的才会有效，而不能是匿名函数。

3 函数对象和其他内部对象的关系

除了函数对象，还有很多内部对象，比如：Object、Array、Date、RegExp、Math、Error。这些名称实际上表示一个类型，可以通过 new 操作符返回一个对象。然而函数对象和其他对象不同，当用 typeof 得到一个函数对象的类型时，它仍然会返回字符串“function”，而 typeof 一个数组对象或其他对象时，它会返回字符串“object”。下面代码表示 typeof 不同类型的情况：

```
alert(typeof(Function));
alert(typeof(new Function()));
alert(typeof(Array));
alert(typeof(Object));
alert(typeof(new Array()));
alert(typeof(new Date()));
alert(typeof(new Object()));
```

运行这段代码可以发现：前面 4 条语句都会显示“function”，而后面 3 条语句则显示“object”，可见新建一个 function 实际上是返回一个函数，这与其他的对象有很大的不同。其他的类型 Array、Object 等都会通过 new 操作符返回一个普通对象。尽管函数本身也是一个对象，但它与普通的对象还是有区别的，因为它同时也是对象构造器，也就是说，可以新建一个函数来返回一个对象，这在前面已经介绍。所有 typeof 返回“function”的对象都是函数对象。也称这样的对象为构造器，因而，所有的构造器都是对象，但不是所有的对象都是构造器。

既然函数本身也是一个对象，它们的类型是 function，联想到 C++、Java 等面向对象语言的类定义，可以猜测到 Function 类型的作用所在，那就是可以给函数对象本身定义一些方法和属性，借助于函数的 prototype 对象，可以很方便地修改和扩充 Function 类型的定义，例如下面扩展了函数类型 Function，为其增加了 method1 方法，作用是弹出对话框显示“function”：

```
Function.prototype.method1=function(){
  alert("function");
}
function func1(a,b,c){
    return a+b+c;
}
func1.method1();
func1.method1.method1();
```

注意最后一个语句：func1.method1.mehotd1 ()，它调用了method1 这个函数对象的 method1 方法。虽然看上去有点容易混淆，但仔细观察一下语法还是很明确的，这是一个递归的定义。因为 method1 本身也是一个函数，所以它同样具有函数对象的属性和方法，所有对 Function 类型的方法扩充都具有这样的递归性质。

Function 是所有函数对象的基础，而 Object 则是所有对象（包括函数对象）的基础。在 JS 中，任何一个对象都是 Object 的实例，因此，可以修改 Object 这个类型来让所有的对象具有一些通用的属性和方法，修改 Object 类型是通过 prototype 来完成的：

```
Object.prototype.getType=function(){
  return typeof(this);
}
var array1=new Array();
function func1(a,b){
  return a+b;
}
alert(array1.getType());
alert(func1.getType());
```

上面的代码为所有的对象添加了 getType 方法，作用是返回该对象的类型。两条 alert 语句分别会显示“object”和“function”。

4 函数作为参数传递

在前面已经介绍了函数对象本质，每个函数都被表示为一个特殊的对象，可以方便地将其赋值给一个变量，再通过这个变量名进行函数调用。作为一个变量，它可以以参数的形式传递给另一个函数，这在前面介绍 JS 事件处理机制中已经看到过这样的用法，例如下面的程序将 func1 作为参数传递给 func2：

```
function func1(theFunc){
  theFunc();
}
function func2(){
  alert("ok");
}
func1(func2);
```

在最后一条语句中，func2 作为一个对象传递给了 func1 的形参 theFunc，再由 func1 内部进行 theFunc 的调用。事实上，将函数作为参数传递，或者是将函数赋值给其他变量是所有事件机制的基础。

例如，若需要在页面载入时进行一些初始化工作，可以先定义一个 init 的初始化函数，再通过 window.onload=init;语句将其绑定到页面载入完成的事件。这里的 init 就是一个函数对象，它可以加入 window 的 onload 事件列表。

5 传递给函数的隐含参数

当进行函数调用时，除了指定的参数外，还创建一个隐含的对象——arguments。arguments 是一个类似数组但不是数组的对象，说它类似是因为它具有与数组一样的访问性质，可以用 arguments [index] 这样的语法取值，拥有数组长度属性 length。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，例如：

```
function func(a,b){
  alert(a);
  alert(b);
  for(var i=0;i<arguments.length;i++){
    alert(arguments[i]);
  }
}
  func(1,2,3);
```

代码运行时会依次显示：1，2，1，2，3。因此，在定义函数的时候，即使不指定参数列表，仍然可以通过 arguments 引用到所获得的参数，这给编程带来了很大的灵活性。

arguments 对象的另一个属性是 callee，它表示对函数对象本身的引用，这有利于实现无名函数的递归或者保证函数的封装性，例如使用递归来计算 1 到 n 的自然数之和：

```
var sum=function(n){
  if(1==n)return 1;
  else return n+sum(n-1);
}
  alert(sum(100));
```

其中函数内部包含了对 sum 自身的调用，然而对于 JS 来说，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，不能很好地体现出是调用自身，所以使用 arguments.callee 属性会是一个较好的办法：

```
var sum=function(n){
  if(1==n)return 1;
  else return n+arguments.callee(n-1);
}
  alert(sum(100));
```

callee 属性并不是 arguments 不同于数组对象的惟一特征，下面的代码说明了 arguments 不是由 Array 类型创建：

```
Array.prototype.p1=1;
alert(new Array().p1);
function func(){
  alert(arguments.p1);
}
  func();
```

运行代码可以发现，第一个 alert 语句显示为 1，即表示数组对象拥有属性 p1，而 func 调用则显示为“undefined”，即 p1

不是 arguments 的属性，由此可见，arguments 并不是一个数组对象。

6 函数的 apply、call 方法和 length 属性

JavaScript 为函数对象定义了两个方法：apply 和 call，它们的作用都是将函数绑定到另外一个对象上去运行，两者仅在定义参数的方式上有所区别：

```
Function.prototype.apply(thisArg,argArray);
Function.prototype.call(thisArg[,arg1[,arg2…]]);
```

从函数原型可以看到，第一个参数都被取名为 thisArg，即所有函数内部的 this 指针都会被赋值为 thisArg，这就实现了将函数作为另外一个对象运行的目的。两个方法除了 thisArg 参数，其余都是为 Function 对象传递的参数。下面的代码说明了 apply 和 call 方法的工作方式：

```
//定义一个函数 func1,具有属性 p 和方法 A
function func1(){
    this.p="func1-";
    this.A=function(arg){
        alert(this.p+arg);
    }
}
//定义一个函数 func2,具有属性 p 和方法 B
function func2(){
    this.p="func2-";
    this.B=function(arg){
        alert(this.p+arg);
    }
}
var obj1=new func1();
var obj2=new func2();
obj1.A("byA");   //显示 func1-byA
obj2.B("byB");   //显示 func2-byB
obj1.A.apply(obj2,["byA"]); //显示 func2-byA,其中["byA"]是仅
//有一个元素的数组,下同
obj2.B.apply(obj1,["byB"]); //显示 func1-byB
obj1.A.call(obj2,"byA"); //显示 func2-byA
obj2.B.call(obj1,"byB"); //显示 func1-byB
```

可以看出，obj1 的方法 A 被绑定到 obj2 运行后，整个函数 A 的运行环境就转移到了 obj2，即 this 指针指向了 obj2。同样 obj2 的函数 B 也可以绑定到 obj1 对象去运行。代码的最后 4 行显示了 apply 和 call 函数参数形式的区别。

与 arguments 的 length 属性不同，函数对象还有一个属性 length，它表示函数定义时所指定参数的个数，而非调用时实际传递的参数个数。下面的代码将显示 2：

```
function sum(a,b){
    return a+b;
}
    alert(sum.length);
```

7 JavaScript 中的 this 指针

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。

和其他面向对象的语言不同，JS 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，在前面讲函数的 apply 和 call 方法时已经有过这样的例子。为了方便理解，再来看下面的示例：

```
<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，这是由 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，也可以更好地理解 JS 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属于一个集合，而是可以动态地属于多个集合。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

（下转第 25 页）

中有更详细的描述，有兴趣的读者可以参考相关文档说明。

此时，来测试菜单的显示效果，下面的语句比较简单，关于 clearLcd () 函数，调用前面实现的函数可以很方便地完成，具体的命令方式可以参考 sed1335 文档。程序全速运行后，实现的菜单效果见图 3 所示：

```
void main()
{
  initialize();     //初始化
  clearLcd();       //清屏函数
  menuTest();       //显示菜单
  while(1)
  {
  }
}
```

图 3　菜单显示效果

2.3　补充说明

在上面的介绍过程中，可以看到，程序中有 AdvWriteString (2,8," 1" ,NOR) 这样的代码，其功能是：完成字符（数字，字母，特殊符号等）的显示，处理方法与写汉字时基本类似，只需注意写入的数据宽度。

在处理汉字时，采用机械取字模方式，这样在菜单信息固定且提示信息不多的情况下，可以节省字模数据占用的存储空间，相对而言也比较灵活。当然，也可以将常用汉字库（含一及，二级）的所有字模数据，预先存到 Flash 中，然后，根据区位码到内码的转化关系，在程序中作处理，也可以达到同样的效果。

3　结语

至此，详细讲解了基于 SED1335 控制器的 LCD 显示模块在 At91sam7s256 上实现的详细步骤，并在最后部分对汉字处理的方法改进做了简要说明。就目前来说，虽然 TFT 屏应用广泛，但在中低端嵌入式设备上，文中介绍的这类点阵 LCD 仍有比较广泛的应用。

参考文献

[1] SED1335 控制器图形液晶显示模块使用手册 . VP . 2001.
[2] Atmel7s256 数据手册. Atmel. 2007.

(收稿日期：2009-3-22)

(上接第 16 页)

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 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 JS 进行面向对象程序设计的一个技巧。

```
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。

```
   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。

8　结语

可以看出，JavaScript 将函数作为一个数据类型来处理，既是对象，定义的函数名也是一个变量，这给 JavaScript 的函数带来一些新的特性，增加了编程的灵活性。

参考文献

[1] 祝红涛. Ajax 从入门到精通. 电子工业出版社，2008.
[2] 褚法政. Ajax 开发技术原理与实践教程，2007.
[3] JavaScript 参考教程.http://www.iselong.com/online/ebooks/javascript/.

(收稿日期：2009-3-26)

JavaScript中函数与对象的解析

作者： 薛晓冬
作者单位：
刊名： 电脑编程技巧与维护
英文刊名： COMPUTER PROGRAMMING SKILLS & MAINTENANCE
年，卷(期)： 2009(11)

参考文献(3条)

1. JavaScript参考教程

2. 褚法政 Ajax开发技术原理与实践教程 2007

3. 祝红涛 Ajax从入门到精通 2008