奇妙的 JavaScript 函数

一、概述

不得不说 JavaScript 是一门让人又爱又恨的语言，她的语法设计中有多少让人爱不释手的，就有多少让人无语抓狂的。今天咱们就来看看 JavaScript 函数中的奇葩语法。

二、函数也是对象

啥？函数也是对象？这在很多其他编程语言中是一件非常不可理解的事情，函数是一段可以重复执行的代码块，为代码复用而生，它怎么能成一个对象呢？这一点我们无从解释，JavaScript 中就是这样做的。我们来看几个证据：

1.证据 1：函数是引用类型

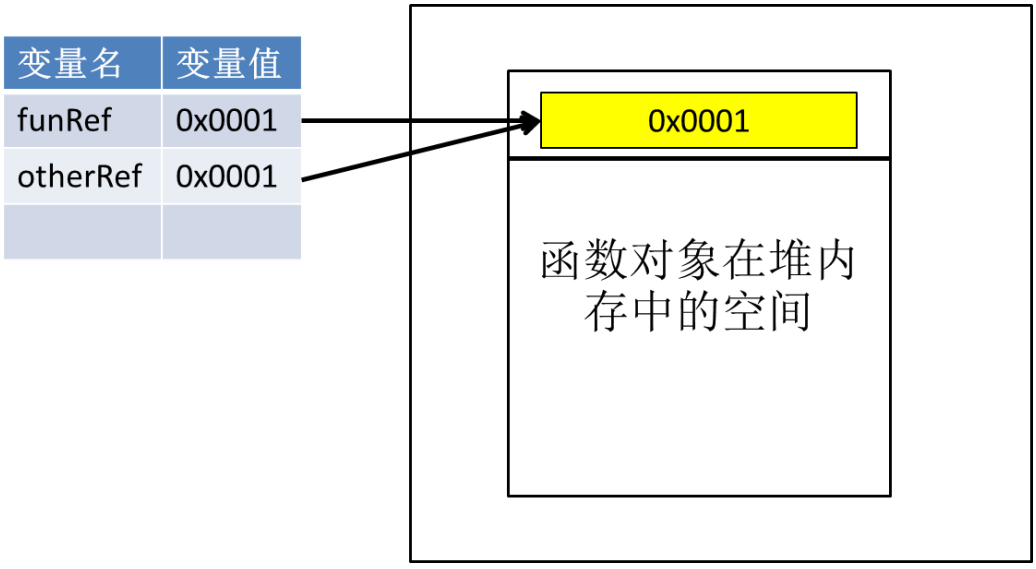
```
//声明一个函数
var funRef = function(){};

//将函数的引用赋值给另一个变量
var otherRef = funRef;

//通过原来的引用给函数对象添加一个属性
funRef.newProperty = "NewValue";

//通过新的引用访问新添加的属性
console.log(otherRef.newProperty);
```

结果是能够得到 NewValue 这个值的，原因很简单，这两个引用指向的是同一块内存空间。



2.证据 2：函数对象也是由构造器函数创建的

在 JavaScript 中任何一个对象都有 constructor 属性，指向创建这个对象的构造器函数，这里就不举例了。那么函数对象有没有创建它的构造器函数呢？有！

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

console.log('函数对象 sum 的构造器函数是：'+sum.constructor);
```

执行结果：

函数对象 sum 的构造器函数是：function Function() { [native code] }

和普通对象一样，函数对象也是构造器创建的呀！所以人家当然是对象！

三、函数对象的妙用

1.携带“静态工具方法”

在 Java 等强类型语言中，工具方法往往被声明为静态形式，以便于使用类名直接访问。而 JavaScript 中没有静态与非静态的区别，在函数对象上添加了方法就可以通过函数名直接访问进而调用。

```
    function hasTool() {}

    hasTool.toolFunction = function(){
        return "工具方法~";
    };

    console.log(hasTool.toolFunction());
```

这一点在 jQuery 中有非常明显的体现：

2.以不同形式引用函数对象

正因为函数也是对象，所以函数就可以堂而皇之的作为一个数据被放在任何地方。又因为在 JavaScript 中调用一个函数只需要在函数的引用后面加上()即可，所以调用函数的方式可谓千变万化！

①赋值给其他变量

```
var otherRef = sum;
result = otherRef(5,8);
console.log("otherRef(5,8)="+result);
```

②赋值给对象属性

```
var obj = {
    myName:"Tom",
    myFun:sum
};

result= obj.myFun(4, 6);
console.log("obj.myFun(4, 6)="+result);
```

③数组元素

```
var arr = ["hello",true,30,otherRef];
result = arr[3](3,3);
console.log("arr[3](3,3)="+result);
```

⑤函数的返回值

```
function returnSum() {
    return function(){
        return "Happy message";
    };
}

result = returnSum()();
console.log("returnSum()()="+result);
```

2.闭包

在 JavaScript 中子作用域可以访问父作用域中的变量，但父作用域不可以访问子作用域中的变量，此时每一个子作用域就可以看做一个闭包，而函数往往是最典型的闭包。那么如何将一个变量的作用域从函数中延伸到函数外呢？很简单，在父函数中将一个使用了父函数变量的子函数的引用作为父函数返回值返回即可。

```
//包含特殊数据的一个函数
function variableFunction() {

```

```
        var variableForGloabl = "需要在上一级作用域中使用的变量";

        //声明一个子函数，这个函数是可以访问到variableForGloabl的
        function getVariable() {
            return variableForGloabl;
        }

        //将子函数的引用返回，这一点很关键
        return getVariable;

    }

    //获取getVariable()函数的引用
    var funRef = variableFunction();

    //调用有权访问局部变量的getVariable()函数获取数据
    var data = funRef();
    console.log("data="+data);

    //可以多次调用获取
    data = funRef();
    console.log("data="+data);
```

闭包的详细内容请参照《尚硅谷_JavaScript 闭包.docx》

3.回调函数

在很多编程语言中都有回调函数的概念，回调函数指的是函数声明后自己不调用，而是交给系统或其他函数调用。JavaScript 中由于函数是对象，所以要实现回调只需要将函数的引用交给系统或指定函数即可。

```
//由系统调用的典型回调函数
btnEle.onclick = function(){
    alert("Hello!");
};
```

四、参数

在 JavaScript 中调用一个函数，允许传入的实参与形参不符。具体说就是：不检查参数的个数和类型，所以 JavaScript 中函数的使用非常的灵活。

1.不检查实参列表

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

```
    var result = sum(2,3);
    console.log("result="+result);//5

    result = sum(2,3,4);
    console.log("result="+result);//5

    result = sum(2);
    console.log("result="+result);//NaN表示这个结果不是一个数字，但不报错

    result = sum("a","b");
    console.log("result="+result);//ab
```

2.可变参数

由于 JavaScript 不检查实参列表，所以 JavaScript 中的函数天生就支持可变参数，我们可以利用这一点实现非常高效的代码编写。

①形参个数

函数名.length 可以返回函数声明时指定了几个形参

```
    function sum(a,b,c) {
        return a+b+c;
    }

    var argumentArr = [];
    for(var i = 0; i < sum.length; i++) {
        argumentArr[i] = i+5;
    }
    var result = sum(argumentArr[0], argumentArr[1], argumentArr[2]);
    console.log("result="+result);
```

执行结果：
result=18

②实参数组

在 JavaScript 函数内部有一个 arguments 对象可以直接使用，严格来说这个对象并不是数组，但可以当做数组来用，所以通常称为伪数组。这个数组中包含了调用函数时传入的全部实参。

```
function superSum() {
    var count = 0;
```

```
    for(var i = 0; i < arguments.length; i++) {
        var number = arguments[i];
        count = count + number;
    }
    return count;
}

result = superSum(1,2);
console.log("superSum(1,2)="+result);

result = superSum(1,2,3);
console.log("superSum(1,2,3)="+result);
```

五、call()和 apply()

在 Java 中，一个函数是不可能被任何一个对象调用的，但在 JavaScript 中就可以。借助函数对象自身的 call()和 apply()函数，函数可以被任意对象调用。

1.call()

call()函数的用法是：函数对象.call(目标对象,参数列表)

```
    var wuDa = {
        name : "武大",
        showName : function(doWhat) {
            //函数中的this引用的是调用这个函数的对象
            console.log(this.name + "和潘金莲"+doWhat);
        }
    };

    wuDa.showName("卖炊饼");

    var xiMen = {
        name : "西门"
    };

    //在JavaScript中，函数可以被其他对象调用
    //函数引用.call(调用函数的对象,函数执行时需要的参数);
    wuDa.showName.call(xiMen, "去马尔代夫看星星");
```

2.apply()

和 call()的区别仅仅是要求传入参数要求是数组类型

```
    var wuDa = {
        name : "武大",
        showName : function (wife){
```

```
            console.log(this.name + "和" + wife);
        }
    };

    var xiMen = {
        name : "西门"
    };

    //函数对象的apply()方法，与call()函数的区别是传入的参数要求是数组类型
    wuDa.showName.apply(xiMen, ["小潘潘~~~"]);
```

3.实际意义

call()和 apply()在 JavaScript 中非常广泛的用途。

①遍历数组

```
/*
自定义的遍历数组的函数
arr参数：要遍历的数组
callBack：让每一个数组元素都调用的回调函数
*/
function myEach(arr,callBack) {
    for(var i = 0; i < arr.length; i++) {
        //遍历得到数组元素
        var ele = arr[i];
        //让数组元素调用回调函数
        callBack.call(ele);
    }
}
```

②参与实现继承

JavaScript 中并没有类的概念，所以面向对象程序设计中的很多概念都只能模拟而没有直接的支持，“继承”就是如此。

```
    function Person(name,age) {
        this.name = name;
        this.age = age;
    }

    //如何借助已有的Person函数简化Student函数中的重复代码？
    function Student(name,age,subject) {
        //this.name = name;
        //this.age = age;
```

```
        //借助call()函数，让Student对象调用Person()函数，这样Person()函数内部的this引用的就是Student对象
        Person.call(this,name,age);
        this.subject = subject;
    }

    //合并“子类”和“父类”的原型对象，让Student的对象是Person的实例
    Student.prototype = Person.prototype;
```