JavaScript 函数参数 JavaScript 闭包

JavaScript 函数调用

JavaScript 函数有 4 种调用方式。

每种方式的不同在于 **this** 的初始化。

this 关键字

一般而言，在Javascript中，this指向函数执行时的当前对象。

注意 **this** 是保留关键字，你不能修改 **this** 的值。

调用 JavaScript 函数

在之前的章节中我们已经学会了如何创建函数。

函数中的代码在函数被调用后执行。

作为一个函数调用

实例

```
function myFunction(a, b) {
return a * b;
}
myFunction(10, 2); // myFunction(10, 2) 返回 20
```

尝试一下 »

以上函数不属于任何对象。但是在 JavaScript 中它始终是默认的全局对象。

在 HTML 中默认的全局对象是 HTML 页面本身，所以函数是属于 HTML 页面。

在浏览器中的页面对象是浏览器窗口(window 对象)。以上函数会自动变为 window 对象的函数。

myFunction() 和 window.myFunction() 是一样的：

实例

```
function myFunction(a, b) {
return a * b;
}
window.myFunction(10, 2); // window.myFunction(10, 2) 返回 20
```

尝试一下 »

这是调用 JavaScript 函数常用的方法， 但不是良好的编程习惯
全局变量，方法或函数容易造成命名冲突的bug。

全局对象

当函数没有被自身的对象调用 时 **this** 的值就会变成全局对象。

在 web 浏览器中全局对象是浏览器窗口（window 对象）。

该实例返回 **this** 的值是 window 对象:

实例

```
function myFunction() {
return this;
}
myFunction(); // 返回 window 对象
```

尝试一下 »

函数作为全局对象调用，会使 **this** 的值成为全局对象。
使用 window 对象作为一个变量容易造成程序崩溃。

函数作为方法调用

在 JavaScript 中你可以将函数定义为对象的方法。

以下实例创建了一个对象 (**myObject**), 对象有两个属性 (**firstName** 和 **lastName**), 及一个方法 (**fullName**):

实例

```
var myObject = {
firstName:"John",
lastName: "Doe",
fullName: function () {
return this.firstName + " " + this.lastName;
}
}
myObject.fullName(); // 返回 "John Doe"
```

尝试一下 »

fullName 方法是一个函数。函数属于对象。 **myObject** 是函数的所有者。

this对象，拥有 JavaScript 代码。实例中 **this** 的值为 **myObject** 对象。

测试以下！修改 **fullName** 方法并返回 **this** 值:

实例

```
var myObject = {
firstName:"John",
lastName: "Doe",
fullName: function () {
return this;
}
}
myObject.fullName(); // 返回 [object Object] (所有者对象)
```

尝试一下 »

函数作为对象方法调用，会使得 **this** 的值成为对象本身。

使用构造函数调用函数

如果函数调用前使用了 **new** 关键字, 则是调用了构造函数。

这看起来就像创建了新的函数，但实际上 JavaScript 函数是重新创建的对象：

实例

```
// 构造函数:
function myFunction(arg1, arg2) {
this.firstName = arg1;
this.lastName = arg2;
}
// This creates a new object
var x = new myFunction("John","Doe");
x.firstName; // 返回 "John"
```

尝试一下 »

构造函数的调用会创建一个新的对象。新对象会继承构造函数的属性和方法。

构造函数中 **this** 关键字没有任何的值。
this 的值在函数调用实例化对象(new object)时创建。

作为函数方法调用函数

在 JavaScript 中, 函数是对象。JavaScript 函数有它的属性和方法。

call() 和 **apply()** 是预定义的函数方法。 两个方法可用于调用函数，两个方法的第一个参数必须是对象本身。

实例

```
function myFunction(a, b) {
return a * b;
}
myObject = myFunction.call(myObject, 10, 2); // 返回 20
```

尝试一下 »

实例

```
function myFunction(a, b) {
return a * b;
}
myArray = [10, 2];
myObject = myFunction.apply(myObject, myArray); // 返回 20
```

尝试一下 »

两个方法都使用了对象本身作为第一个参数。 两者的区别在于第二个参数： apply传入的是一个参数数组，也就是将多个参数组合成为一个数组传入，而call则作为call的参数传入（从第二个参数开始）。

在 JavaScript 严格模式(strict mode)下, 在调用函数时第一个参数会成为 **this** 的值， 即使该参数不是一个对象。

在 JavaScript 非严格模式(non-strict mode)下, 如果第一个参数的值是 null 或 undefined, 它将使用全局对象替代。

通过 call() 或 apply() 方法你可以设置 **this** 的值, 且作为已存在对象的新方法调用。

JavaScript 函数参数 JavaScript 闭包

2 篇笔记

写笔记

this 是 JavaScript 语言的一个关键字。

它代表函数运行时，自动生成的一个内部对象，只能在函数内部使用。比如：

```
function test() {
    this.x = 1;
}
```

随着函数使用场合的不同，this 的值会发生变化。但是有一个总的原则，那就是this指的是，调用函数的那个对象。

WooKong 2年前 (2017-05-26)

作为函数方法调用函数时，此函数执行了相当于 java 中静态函数的功能。

```
<script>
var myObject, myArray;
myObject={
    name: "hahaha ",
    hsk: "en"
};
function myFunction(a, b) {
    alert(this);
    return this.name +this.hsk;
}
myArray = [10, 2]
myObject = myFunction.apply(myObject, myArray);
document.getElementById("demo").innerHTML = myObject;
</script>
```

尝试一下 »

可以用的频率较高的函数作这样的设置，为对象执行相关操作。

ha oh 6个月前 (09-23)