<http://www.cnblogs.com/fsjohnhuang/p/4160942.html>

**一、前言**

大家先预计一下以下四个函数调用的结果吧！

[复制代码](javascript:void(0);)

var test = function(){

console.log('hello world')  
 return 'fsjohnhuang'

}  
test.call() // ①

Function.prototype.call(test) // ②

Function.prototype.call.call(test) // ③

Function.prototype.call.call(Function.prototype.call, test) // ④

[复制代码](javascript:void(0);)

  揭晓：①、③和④. 控制台显示hello world，并返回fsjohnhuang。②. 返回undefined且不会调用test函数；

  那到底是啥回事呢？下面将一一道来。

**二、从常用的call函数说起**

  还是通过代码说事吧

var test2 = function(){

console.log(this)

return 'fsjohnhuang'

}

test2() // 控制台显示window对象信息，返回值为fsjohnhuang

test2.call({msg: 'hello world'}) // 控制台显示{msg:'hello world'}对象信息，返回值为fsjohnhuang

  test2.call实际上是调用 Function.prototype.call (thisArg [ , arg1 [ , arg2, … ] ] ) ，而其作用我想大家都了解的，但其内部的工作原理是怎样的呢？ 这时我们可以参考[ECMAScript5.1语言规范](http://es5.github.io/#x15.3.4.4)。以下是参照规范的伪代码（各浏览器的具体实现均不尽相同）

[复制代码](javascript:void(0);)

Function.prototype.call = function(thisArg, arg1, arg2, ...) {

/\*\*\* 注意：this指向调用call的那个对象或函数 \*\*\*/

// 1. 调用内部的IsCallable(this)检查是否可调用，返回false则抛TypeError

if (![[IsCallable]](this)) throw new TypeError()

// 2. 创建一个空列表

// 3. 将arg1及后面的入参保存到argList中

var argList = [].slice.call(arguments, 1)

// 4. 调用内部的[[Call]]函数

return [[Call]](this, thisArg, argList)

}

[复制代码](javascript:void(0);)

  那现在我们可以分析一下 ①test.call() ，并以其为基础去理解后续的内容。它内部实现的伪代码如下：

test.call = function(thisArg, arg1, arg2, ...){

if (![[IsCallable]](test)) throw new TypeError()

var argList = [].slice.call(arguments, 1)

return [[Call]](test, thisArg, argList)

}

  下面我们再来分析② Function.prototype.call(test) ，伪代码如下：

[复制代码](javascript:void(0);)

Function.prototype.call = function(test, arg1, arg2, ...){

/\*\*\* Function.prototype是一个function Empty(){}函数 \*\*\*/

if (![[IsCallable]](Function.prototype)) throw new TypeError()

var argList = [].slice.call(arguments, 1)

// 实际上就是调用Empty函数而已，那返回undefined是理所当然的

return [[Call]](Function.prototype, test, argList)

}

[复制代码](javascript:void(0);)

**三、Function.prototype.call.call内部究竟又干嘛了？**

  有了上面的基础那么Function.prototype.call.call就不难理解了。就是以最后一个call函数的thisArg作为Function.prototype.call的this值啦！伪代码如下：

[复制代码](javascript:void(0);)

// test作为thisArg传入

Function.prototype.call.call = function(test, arg1, arg2,...){

if ([[IsCallable]](Function.prototype.call)) throw new TypeError()

var argList = [].slice.call(arguments, 1)

return [[Call]](Function.prototype.call, test, argList)

}

// test作为函数的this值

// 注意：入参thisArg的值为Function.prototype.call.call的入参arg1

Function.prototype.call = function(thisArg, arg1, arg2,...){

if ([[IsCallable]](test)) throw new TypeError()

var argList = [].slice.call(arguments, 1)

return [[Call]](test, thisArg, argList)

}

[复制代码](javascript:void(0);)

**四、见鬼的合体技——Function.prototype.call.call(Function.prototype.call, test)**

  看伪代码理解吧！

[复制代码](javascript:void(0);)

// test作为arg1传入

Function.prototype.call.call = function(Function.prototype.call, test){

if ([[IsCallable]](Function.prototype.call)) throw new TypeError()

var argList = [].slice.call(arguments, 1)

return [[Call]](Function.prototype.call, Function.prototype.call, argList)

}

Function.prototype.call = function(test){

if ([[IsCallable]](Function.prototype.call)) throw new TypeError()

var argList = [].slice.call(arguments, 1)

return [[Call]](Function.prototype.call, test, argList)

}

Function.prototype.call = function(thisArg){

if ([[IsCallable]](test)) throw new TypeError()

var argList = [].slice.call(arguments, 1)

return [[Call]](test, thisArg, argList)

}

[复制代码](javascript:void(0);)

  这种合体技不就是比第三节的多了一个步吗？有必有吗？

**五、新玩法——遍历执行函数数组**

[复制代码](javascript:void(0);)

Array.prototype.resolve = function(){

　　this.forEach(Function.prototype.call, Function.prototype.call)

}

var cbs = [function(){console.log(1)}, function(){console.log(2)}]

cbs.resolve()

// 控制台输出

// 1

// 2

[复制代码](javascript:void(0);)

   这是为什么呢？那先要看看 Array.prototype.forEach(fn, thisArg) 的内部实现了，伪代码如下：

[复制代码](javascript:void(0);)

Array.prototype.forEach = function(fn, thisArg){

var item

for (var i = 0, len = this.length; i < len; ++i){

item = this[i]

fn.call(thisArg, item, i, this)

}

}

[复制代码](javascript:void(0);)