[立即调用的函数表达式](http://www.cnblogs.com/TomXu/archive/2011/12/31/2289423.html)

**前言**

大家学JavaScript的时候，经常遇到自执行匿名函数的代码，今天我们主要就来想想说一下自执行。  
  
在详细了解这个之前，我们来谈了解一下“自执行”这个叫法，本文对这个功能的叫法也不一定完全对，主要是看个人如何理解，因为有的人说立即调用，有的人说自动执行，所以你完全可以按照你自己的理解来取一个名字，不过我听很多人都叫它为“自执行”，但作者后面说了很多，来说服大家称呼为“立即调用的函数表达式”。

本文英文原文地址：http://benalman.com/news/2010/11/immediately-invoked-function-expression/

**什么是自执行？**

在JavaScript里，任何function在执行的时候都会创建一个执行上下文，因为为function声明的变量和function有可能只在该function内部，这个上下文，在调用function的时候，提供了一种简单的方式来创建自由变量或私有子function。

// 由于该function里返回了另外一个function，其中这个function可以访问自由变量i  
// 所有说，这个内部的function实际上是有权限可以调用内部的对象。  
  
function makeCounter() {  
 // 只能在makeCounter内部访问i  
 var i = 0;  
  
 return function () {  
 console.log(++i);  
 };  
}  
  
// 注意，counter和counter2是不同的实例，分别有自己范围内的i。  
  
var counter = makeCounter();  
counter(); // logs: 1  
counter(); // logs: 2  
  
var counter2 = makeCounter();  
counter2(); // logs: 1  
counter2(); // logs: 2  
  
alert(i); // 引用错误：i没有defind（因为i是存在于makeCounter内部）。

很多情况下，我们不需要makeCounter多个实例，甚至某些case下，我们也不需要显示的返回值，OK，往下看。

**问题的核心**

当你声明类似function foo(){}或var foo = function(){}函数的时候，通过在后面加个括弧就可以实现自执行，例如foo()，看代码：

// 因为想下面第一个声明的function可以在后面加一个括弧()就可以自己执行了，比如foo()，  
// 因为foo仅仅是function() { /\* code \*/ }这个表达式的一个引用  
   
var foo = function(){ /\* code \*/ }  
   
// ...是不是意味着后面加个括弧都可以自动执行？  
   
function(){ /\* code \*/ }(); // SyntaxError: Unexpected token (  
//

上述代码，如果甚至运行，第2个代码会出错，因为在解析器解析全局的function或者function内部function关键字的时候，默认是认为function声明，而不是function表达式，如果你不显示告诉编译器，它默认会声明成一个缺少名字的function，并且抛出一个语法错误信息，因为function声明需要一个名字。

**旁白：函数(function)，括弧(paren)，语法错误(SyntaxError)**

有趣的是，即便你为上面那个错误的代码加上一个名字，他也会提示语法错误，只不过和上面的原因不一样。在一个表达式后面加上括号()，该表达式会立即执行，但是在一个语句后面加上括号()，是完全不一样的意思，他的只是分组操作符。

// 下面这个function在语法上是没问题的，但是依然只是一个语句  
// 加上括号()以后依然会报错，因为分组操作符需要包含表达式  
   
function foo(){ /\* code \*/ }(); // SyntaxError: Unexpected token )  
   
// 但是如果你在括弧()里传入一个表达式，将不会有异常抛出  
// 但是foo函数依然不会执行  
function foo(){ /\* code \*/ }( 1 );  
   
// 因为它完全等价于下面这个代码，一个function声明后面，又声明了一个毫无关系的表达式：   
function foo(){ /\* code \*/ }  
   
( 1 );

你可以访问[ECMA-262-3 in detail. Chapter 5. Functions](http://dmitrysoshnikov.com/ecmascript/chapter-5-functions/#question-about-surrounding-parentheses) 获取进一步的信息。

**自执行函数表达式**

要解决上述问题，非常简单，我们只需要用大括弧将代码的代码全部括住就行了，因为JavaScript里括弧()里面不能包含语句，所以在这一点上，解析器在解析function关键字的时候，会将相应的代码解析成function表达式，而不是function声明。

// 下面2个括弧()都会立即执行  
  
(function () { /\* code \*/ } ()); // 推荐使用这个  
(function () { /\* code \*/ })(); // 但是这个也是可以用的  
  
// 由于括弧()和JS的&&，异或，逗号等操作符是在函数表达式和函数声明上消除歧义的  
// 所以一旦解析器知道其中一个已经是表达式了，其它的也都默认为表达式了  
// 不过，请注意下一章节的内容解释  
  
var i = function () { return 10; } ();  
true && function () { /\* code \*/ } ();  
0, function () { /\* code \*/ } ();  
  
// 如果你不在意返回值，或者不怕难以阅读  
// 你甚至可以在function前面加一元操作符号  
  
!function () { /\* code \*/ } ();  
~function () { /\* code \*/ } ();  
-function () { /\* code \*/ } ();  
+function () { /\* code \*/ } ();  
  
// 还有一个情况，使用new关键字,也可以用，但我不确定它的效率  
// http://twitter.com/kuvos/status/18209252090847232  
  
new function () { /\* code \*/ }  
new function () { /\* code \*/ } () // 如果需要传递参数，只需要加上括弧()

上面所说的括弧是消除歧义的，其实压根就没必要，因为括弧本来内部本来期望的就是函数表达式，但是我们依然用它，主要是为了方便开发人员阅读，当你让这些已经自动执行的表达式赋值给一个变量的时候，我们看到开头有括弧(，很快就能明白，而不需要将代码拉到最后看看到底有没有加括弧。

**用闭包保存状态**

和普通function执行的时候传参数一样，自执行的函数表达式也可以这么传参，因为闭包直接可以引用传入的这些参数，利用这些被lock住的传入参数，自执行函数表达式可以有效地保存状态。

// 这个代码是错误的，因为变量i从来就没背locked住  
// 相反，当循环执行以后，我们在点击的时候i才获得数值  
// 因为这个时候i操真正获得值  
// 所以说无论点击那个连接，最终显示的都是I am link #10（如果有10个a元素的话）  
  
var elems = document.getElementsByTagName('a');  
  
for (var i = 0; i < elems.length; i++) {  
  
 elems[i].addEventListener('click', function (e) {  
 e.preventDefault();  
 alert('I am link #' + i);  
 }, 'false');  
  
}  
  
// 这个是可以用的，因为他在自执行函数表达式闭包内部  
// i的值作为locked的索引存在，在循环执行结束以后，尽管最后i的值变成了a元素总数（例如10）  
// 但闭包内部的lockedInIndex值是没有改变，因为他已经执行完毕了  
// 所以当点击连接的时候，结果是正确的  
  
var elems = document.getElementsByTagName('a');  
  
for (var i = 0; i < elems.length; i++) {  
  
 (function (lockedInIndex) {  
  
 elems[i].addEventListener('click', function (e) {  
 e.preventDefault();  
 alert('I am link #' + lockedInIndex);  
 }, 'false');  
  
 })(i);  
  
}  
  
// 你也可以像下面这样应用，在处理函数那里使用自执行函数表达式  
// 而不是在addEventListener外部  
// 但是相对来说，上面的代码更具可读性  
  
var elems = document.getElementsByTagName('a');  
  
for (var i = 0; i < elems.length; i++) {  
  
 elems[i].addEventListener('click', (function (lockedInIndex) {  
 return function (e) {  
 e.preventDefault();  
 alert('I am link #' + lockedInIndex);  
 };  
 })(i), 'false');  
  
}

其实，上面2个例子里的lockedInIndex变量，也可以换成i，因为和外面的i不在一个作用于，所以不会出现问题，这也是匿名函数+闭包的威力。

**自执行匿名函数和立即执行的函数表达式区别**

在这篇帖子里，我们一直叫自执行函数，确切的说是自执行匿名函数（Self-executing anonymous function），但英文原文作者一直倡议使用立即调用的函数表达式（Immediately-Invoked Function Expression）这一名称，作者又举了一堆例子来解释，好吧，我们来看看：

// 这是一个自执行的函数，函数内部执行自身，递归  
function foo() { foo(); }  
  
// 这是一个自执行的匿名函数，因为没有标示名称  
// 必须使用arguments.callee属性来执行自己  
var foo = function () { arguments.callee(); };  
  
// 这可能也是一个自执行的匿名函数，仅仅是foo标示名称引用它自身  
// 如果你将foo改变成其它的，你将得到一个used-to-self-execute匿名函数  
var foo = function () { foo(); };  
  
// 有些人叫这个是自执行的匿名函数（即便它不是），因为它没有调用自身，它只是立即执行而已。  
(function () { /\* code \*/ } ());  
  
// 为函数表达式添加一个标示名称，可以方便Debug  
// 但一定命名了，这个函数就不再是匿名的了  
(function foo() { /\* code \*/ } ());  
  
// 立即调用的函数表达式（IIFE）也可以自执行，不过可能不常用罢了  
(function () { arguments.callee(); } ());  
(function foo() { foo(); } ());  
  
// 另外，下面的代码在黑莓5里执行会出错，因为在一个命名的函数表达式里，他的名称是undefined  
// 呵呵，奇怪  
(function foo() { foo(); } ());

希望这里的一些例子，可以让大家明白，什么叫自执行，什么叫立即调用。

*注：arguments.callee在*[*ECMAScript 5 strict mode*](https://developer.mozilla.org/en/JavaScript/Strict_mode#Differences_in_functions)*里被废弃了，所以在这个模式下，其实是不能用的。*

**最后的旁白：Module模式**

在讲到这个立即调用的函数表达式的时候，我又想起来了Module模式，如果你还不熟悉这个模式，我们先来看看代码：

// 创建一个立即调用的匿名函数表达式  
// return一个变量，其中这个变量里包含你要暴露的东西  
// 返回的这个变量将赋值给counter，而不是外面声明的function自身  
  
var counter = (function () {  
 var i = 0;  
  
 return {  
 get: function () {  
 return i;  
 },  
 set: function (val) {  
 i = val;  
 },  
 increment: function () {  
 return ++i;  
 }  
 };  
} ());  
  
// counter是一个带有多个属性的对象，上面的代码对于属性的体现其实是方法  
  
counter.get(); // 0  
counter.set(3);  
counter.increment(); // 4  
counter.increment(); // 5  
  
counter.i; // undefined 因为i不是返回对象的属性  
i; // 引用错误: i 没有定义（因为i只存在于闭包）