彻底理解javascript的回调函数（推荐） - MoltBoy - 博客园

在javascript中回调函数非常重要，它们几乎无处不在。像其他更加传统的编程语言都有回调函数概念，但是非常奇怪的是，完完整整谈论回调函数的在线教程比较少，倒是有一堆关于call()和apply()函数的，或者有一些简短的关于callback的使用示例。

函数也是对象

想弄明白回调函数，首先的清楚地明白函数的规则。在javascript中，函数是比较奇怪的，但它确确实实是对象。确切地说，函数是用Function()构造函数创建的Function对象。Function对象包含一个字符串，字符串包含函数的javascript代码。假如你是从C语言或者java语言转过来的，这也许看起来很奇怪，代码怎么可能是字符串？但是对于javascript来说，这很平常。数据和代码之间的区别是很模糊的。

```
//可以这样创建函数var fn = new Function("arg1", "arg2", "return arg1 * arg2;");fn(2, 3);   //6
```

这样做的一个好处，可以传递代码给其他函数，也可以传递正则变量或者对象（因为代码字面上只是对象而已）。

传递函数作为回调

很容易把一个函数作为参数传递。

```
function fn(arg1, arg2, callback){    var num = Math.ceil(Math.random() * (arg1 - arg2) + arg2);
callback(num);    //传递结果}fn(10, 20, function(num){    console.log("Callback called! Num: " + num); });
     //结果为10和20之间的随机数
```

可能这样做看起比较麻烦，甚至有点愚蠢，为何不正常地返回结果？但是当遇上必须使用回调函数之时，你也许就不这样认为了！

别挡道

传统函数以参数形式输入数据，并且使用返回语句返回值。理论上，在函数结尾处有一个return返回语句，结构上就是：一个输入点和一个输出点。这比较容易理解，函数本质上就是输入和输出之间实现过程的映射。

但是，当函数的实现过程非常漫长，你是选择等待函数完成处理，还是使用回调函数进行异步处理呢？这种情况下，使用回调函数变得至关重要，例如：AJAX请求。若是使用回调函数进行处理，代码就可以继续进行其他任务，而无需空等。实际开发中，经常在javascript中使用异步调用，甚至在这里强烈推荐使用！

下面有个更加全面的使用AJAX加载XML文件的示例，并且使用了call()函数，在请求对象（requested object）上下文中调用回调函数。

```
function fn(url, callback){    var httpRequest;       //创建XHR    httpRequest = window.XMLHttpRequest ?
new XMLHttpRequest() :     //针对IE进行功能性检测       window.ActiveXObject ? new
ActiveXObject("Microsoft.XMLHTTP") : undefined;        httpRequest.onreadystatechange = function(){
if(httpRequest.readystate === 4 && httpRequest.status === 200){    //状态判断
callback.call(httpRequest.responseXML);         }    };    httpRequest.open("GET", url);
httpRequest.send();}fn("text.xml", function(){       //调用函数    console.log(this);     //此语句后输
出});console.log("this will run before the above callback.");    //此语句先输出
```

我们请求异步处理，意味着我们开始请求时，就告诉它们完成之时调用我们的函数。在实际情况中，onreadystatechange事件处理程序还得考虑请求失败的情况，这里我们是假设xml文件存在并且能被浏览器成功加载。这个例子中，异步函数分配给了onreadystatechange事件，因此不会立刻执行。

最终，第二个console.log语句先执行，因为回调函数直到请求完成才执行。

上述例子不太易于理解，那看看下面的示例：

```
function foo(){    var a = 10;    returnfunction(){       a *= 2;       return a;          };   }var f
= foo();f(); //return 20.f(); //return 40.
```

函数在外部调用，依然可以访问变量a。这都是因为javascript中的作用域是词法性的。**函数式运行在定义它们的作用域中（上述例子中的foo内部的作用域），而不是运行此函数的作用域中。**只要f被定义在foo中，它就可以访问foo中定义的所有的变量，即便是foo的执行已经结束。因为它的作用域会被保存下来，但也只有返回的那个函数才可以访问这个保存下来的作用域。返回一个内嵌匿名函数是创建闭包最常用的手段。