PHP语言, PHP扩展, Zend引擎相关的研究,技术,新闻分享 – 左手代码 右手诗

28 May 09 Javascript作用域原理 (http://www.laruence.com/2009/05/28/863.html)

- 作者: Laruence (http://www.laruence.com) ((http://www.twitter.com/laruence) (http://t.sina.com/laruence) (http://fusion.google.com/add?feedurl=http://www.laruence.com/feed) (mailto:laruence@yahoo.com.cn))
- 本文地址: http://www.laruence.com/2009/05/28/863.html (http://www.laruence.com/2009/05/28/863.html)
- 转载请注明出处

问题的提出

首先看一个例子:

```
var name = 'laruence';
function echo() {
          alert(name);
          var name = 'eve';
          alert(name);
          alert(age);
}

echo();
```

运行结果是什么呢?

上面的问题, 我相信会有很多人会认为是:

```
laruence
eve
[脚本出错]
```

因为会以为在echo中, 第一次alert的时候, 会取到全局变量name的值, 而第二次值被局部变量name覆盖, 所以第二次alert是'eve'. 而age属性没有定义, 所以脚本会出错.

但其实, 运行结果应该是:

```
undefined
eve
[脚本出错]
```

为什么呢?

JavaScript的作用域链

首先让让我们来看看Javasript(简称JS, 不完全代表JScript)的作用域的原理: JS权威指南中有一句很精辟的描述: "JavaScript中的函数运行在它们被定义的作用域里,而不是它们被执行的作用域里."

为了接下来的知识, 你能顺利理解, 我再提醒一下, 在JS中:"一切皆是对象, 函数也是".

在JS中, 作用域的概念和其他语言差不多, 在每次调用一个函数的时候 , 就会进入一个函数内的作用域, 当从函数返回以后, 就返回调用前的作用域.

JS的语法风格和C/C++类似, 但作用域的实现却和C/C++不同, 并非用"堆栈"方式, 而是使用列表, 具体过程如下(ECMA262中所述):
任何执行上下文时刻的作用域, 都是由作用域链(scope chain, 后面介绍)来实现.
在一个函数被定义的时候, 会将它定义时刻的scope chain链接到这个函数对象的[[scope]]属性.
在一个函数对象被调用的时候, 会创建一个活动对象(也就是一个对象), 然后对于每一个函数的形参, 都命名为该活动对象的命名属性, 然后将这个活动对象做为此时的作用域链(scope chain)最前端, 并将这个函数对象的[[scope]]加入到scope chain中.

看个例子:

```
var func = function(lps, rps){
        var name = 'laruence';
        ........
}
func();
```

在执行func的定义语句的时候, 会创建一个这个函数对象的[[scope]]属性(内部属性,只有JS引擎可以访问, 但FireFox的几个引擎(SpiderMonkey和Rhino)提供了私有属性__parent__来访问它), 并将这个[[scope]]属性, 链接到定义它的作用域链上(后面会详细介绍), 此时因为func定义在全局环境, 所以此时的[[scope]]只是指向全局活动对象window active object.

在调用func的时候, 会创建一个活动对象(假设为aObj, 由JS引擎预编译时刻创建, 后面会介绍)，并创建arguments属性, 然后会给这个对象添加俩个命名属性aObj.lps, aObj.rps; 对于每一个在这个函数中申明的局部变量和函数定义, 都作为该活动对象的同名命名属性.

然后将调用参数赋值给形参数，对于缺少的调用参数，赋值为undefined。

然后将这个活动对象做为scope chain的最前端, 并将func的[[scope]]属性所指向的,定义func时候的顶级活动对象, 加入到scope chain.

有了上面的作用域链, 在发生标识符解析的时候, 就会逆向查询当前scope chain列表的每一个活动对象的属性，如果找到同名的就返回。找不到，那就是这个标识符没有被定义。

注意到, 因为函数对象的[[scope]]属性是在定义一个函数的时候决定的, 而非调用的时候, 所以如下面的例子:

```
var name = 'laruence';
function echo() {
        alert(name);
}

function env() {
        var name = 'eve';
        echo();
}

env();
```

运行结果是:

```
laruence
```

结合上面的知识, 我们来看看下面这个例子:

```
function factory() {
        var name = 'laruence';
        var intro = function(){
                alert('I am ' + name);
        }
        return intro;
}

function app(para){
        var name = para;
        var func = factory();
        func();
}

app('eve');
```

当调用app的时候, scope chain是由: {window活动对象(全局)}->{app的活动对象} 组成.

在刚进入app函数体时, app的活动对象有一个arguments属性, 俩个值为undefined的属性: name和func. 和一个值为'eve'的属性para;

此时的scope chain如下:

```
[[scope chain]] = [
{
        para : 'eve',
        name : undefined,
        func : undefined,
        arguments : []
}, {
        window call object
}
```

```
]
```

当调用进入factory的函数体的时候, 此时的factory的scope chain为:

```
[[scope chain]] = [
{
          name : undefined,
          intor : undefined
}, {
          window call object
}
]
```

注意到, 此时的作用域链中, 并不包含app的活动对象.

在定义intro函数的时候, intro函数的[[scope]]为:

```
[[scope chain]] = [
{
          name : 'laruence',
          intor : undefined
}, {
          window call object
}
]
```

从factory函数返回以后,在app体内调用intor的时候, 发生了标识符解析, 而此时的sope chain是:

```
[[scope chain]] = [
{
          intro call object
}, {
          name : 'laruence',
          intor : undefined
}, {
          window call object
}
]
```

因为scope chain中,并不包含factory活动对象. 所以, name标识符解析的结果应该是factory活动对象中的name属性, 也就是'laruence'.

所以运行结果是:

```
I am laruence
```

现在, 大家对"JavaScript中的函数运行在它们被定义的作用域里,而不是它们被执行的作用域里."这句话, 应该有了个全面的认识了吧?

Javascript的预编译

我们都知道,JS是一种脚本语言, JS的执行过程, 是一种翻译执行的过程.
那么JS的执行中, 有没有类似编译的过程呢?

首先, 我们来看一个例子:

```
          <script>
          alert(typeof eve); //function
                    function eve() {
                              alert('I am Laruence');
                    };
          </script>
```

诶? 在alert的时候, eve不是应该还是未定义的么? 怎么eve的类型还是function呢?

恩, 对, 在JS中, 是有预编译的过程的, JS在执行每一段JS代码之前, 都会首先处理var关键字和function定义式(函数定义式和函数表达式).
如上文所说, 在调用函数执行之前, 会首先创建一个活动对象, 然后搜寻这个函数中的局部变量定义,和函数定义, 将变量名和函数名都做为这个活动对象的同名属性, 对于局部变量定义,变量的值会在真正执行的时候才计算, 此时只是简单的赋为undefined.

而对于函数的定义,是一个要注意的地方:

```
<script>
          alert(typeof eve); //结果:function
          alert(typeof walle); //结果:undefined
```

```
        function eve() { //函数定义式
                alert('I am Laruence');
        };
        var walle = function() { //函数表达式
        }
        alert(typeof walle); //结果:function
</script>
```

这就是函数定义式和函数表达式的不同, 对于函数定义式, 会将函数定义提前. 而函数表达式, 会在执行过程中才计算.

说到这里, 顺便说一个问题 :

```
        var name = 'laruence';
        age = 26;
```

我们都知道不使用var关键字定义的变量, 相当于是全局变量, 联系到我们刚才的知识:

在对age做标识符解析的时候, 因为是写操作, 所以当找到到全局的window活动对象的时候都没有找到这个标识符的时候, 会在window活动对象的基础上, 返回一个值为undefined的age属性.

也就是说, age会被定义在顶级作用域中.

现在, 也许你注意到了我刚才说的: JS在执行每一段**JS**代码..
对, 让我们看看下面的例子:

```
<script>
        alert(typeof eve); //结果:undefined
</script>
<script>
        function eve() {
                alert('I am Laruence');
        }
</script>
```

明白了么? 也就是JS的预编译是以段为处理单元的...

揭开谜底

现在让我们回到我们的第一个问题:

当echo函数被调用的时候, echo的活动对象已经被预编译过程创建, 此时echo的活动对象为:

```
[callObj] = {
name : undefined
}
```

当第一次alert的时候, 发生了标识符解析, 在echo的活动对象中找到了name属性, 所以这个name属性, 完全的遮挡了全局活动对象中的name属性.

现在你明白了吧?

Related Posts:

- 2012年1月全球www网站技术报告 (http://www.laruence.com/2012/01/07/2453.html)
- 如何获取一个变量的名字 (http://www.laruence.com/2010/12/08/1716.html)
- Javascript原型链和原型的一个误区 (http://www.laruence.com/2010/05/13/1462.html)
- IE下var的重要性的又一佐证 (http://www.laruence.com/2010/01/21/1254.html)
- 关于Javascript的俩个有趣的探讨 (http://www.laruence.com/2009/09/27/1123.html)
- 深入理解JavaScript定时机制 (http://www.laruence.com/2009/09/23/1089.html)
- 深入理解Javascript之this关键字 (http://www.laruence.com/2009/09/08/1076.html)
- 正确使用JS中的正则 (http://www.laruence.com/2009/08/09/1036.html)
- 使用JS做文档处理 (http://www.laruence.com/2009/05/18/809.html)
- 关于事件模拟 (http://www.laruence.com/2009/05/17/815.html)

Tags: javascript (http://www.laruence.com/tag/javascript) , javascript预编译 (http://www.laruence.com/tag/javascript%e9%a2%84%e7%bc%96%e8%af%91) , scope chain (http://www.laruence.com/tag/scope-chain) , 作用域 (http://www.laruence.com/tag/%e4%bd%9c%e7%94%a8%e5%9f%9f)

Filed in Js/CSS (http://www.laruence.com/category/jscss)

« PHP+Gtk实例(求24点) (http://www.laruence.com/2009/05/26/871.html)
PHP受locale影响的函数 (http://www.laruence.com/2009/05/31/889.html) »

69 Responses to “Javascript作用域原理”

Pages: **[2]** 1 (http://www.laruence.com/?p=863&cp=1#comments) » (http://www.laruence.com/?p=863&cp=1#comments) Show All (http://www.laruence.com/?p=863&cp=all#comments)

geesung (http://none) | 01 Apr 2016 09:20

通俗易懂，好文好文！

phping (http://phping.sinaapp.com/) | 24 Mar 2016 15:48

写得不错，有些地方一时还是难以消化，在实际开发中慢慢消化吧。

Javascript 的“块作用域” – LET | 百作坊 (http://blog.100dos.com/about-javascript-block-scope-let-keyword/) | 02 Nov 2015 09:44

[...] Javascript 是一种脚本语言，在脚本执行时，是要经过预编译的。关于“JavaScript的作用域链”和“Javascript的预编译”可以学习鸟哥的Javascript作用域原理。 [...]

潮有起落 (http://下面的问题已明白) | 27 Aug 2015 12:45

下面的问题已明白，衷心感谢。

潮有起落 (http://有一个问起请教) | 27 Aug 2015 12:26

```
func(){
name=”one”;
alert(name);
}
```

这个变量name会在那个活动的对象中创建同名属性？

javascript作用域链详解 — 好JSER (http://hao.jser.com/archive/8244/) | 21 Aug 2015 05:50

[...] 文章部分实例和内容来自鸟哥的blogJavascript作用域原理 [...]

闭包里的微观世界 — 好JSER (http://hao.jser.com/archive/8132/) | 27 Jul 2015 06:10

[...] 具体内容参见：鸟哥：Javascript作用域和作用域链 [...]

木心 | 04 Jun 2015 23:48

因为scope chain中,并不包含factory活动对象. 所以, name标识符解析的结果应该是factory活动对象中的name属性, 也就是’laruence’.

//这里这一句话是不是打错了哦 应该是scope chain中并不包含app活动对象.所以,name标识符解析的结果应该是factory活动对象的name属性,也就是’laruence’.

还是受教了！
如果是我理解错了请指正！

JavaScript作用域链 — 好JSER (http://hao.jser.com/archive/7839/) | 25 May 2015 15:03

[...] P73 4.2 执行环境及作用域鸟哥：JavaScript作用域原理JavaScript 开发进阶：理解 JavaScript [...]

JavaScript作用域链与闭包 – 吴化吉 (http://wuhuaji.me/?p=44) | 22 May 2015 23:31

[...] 这里引用鸟哥的描述 在调用函数执行之前, 会首先创建一个活动对象, 然后搜寻这个函数中的局部变量定义,和函数定义, 将变量名和函数名都做为这个活动对象的同名属性, 对于局部变量定义,变量的值会在真正执行的时候才计算, 此时只是简单的赋为undefined. [...]

jzz15 | 18 May 2015 10:05

写的真的太好了！！！我看完JavaScript高级程序设计那本书里面的作用域链之后再来看这个感觉更加巩固了理解.

jzz | 18 May 2015 10:04

写的真的太好了！！！我看完JavaScript高级程序设计那本书里面的作用域链之后再来看这个感觉更加巩固了理解.

百度Web前端技术学院编码挑战（TASK 0003） | Anotherhome (https://www.anotherhome.net/1989) | 14 May 2015 21:16

[...] 1. JavaScript作用域 （参考 鸟哥：Javascript作用域原理 理解 JavaScript 作用域和作用域链） [...]

wintercoder (http://www.wintercoder.com) | 12 May 2015 10:54

感谢，看了几遍收获挺大~

javascript学习笔记(1) — 好JSER (http://hao.jser.com/archive/7766/) | 09 May 2015 02:42

[...] 1.鸟哥：Javascript作用域原理 2.理解 JavaScript 作用域和作用域链 [...]

JavaScript作用域和作用域链 - Wenci (http://zengxuebing.com/wordpress/?p=139) | 01 May 2015 16:07

[...] 参考文章：鸟哥:Javascript作用域原理&&理解 JavaScript 作用域和作用域链 [...]

Eric Miao (http://blog.miaozhaofeng.cn) | 04 Mar 2015 15:19

引用：
在定义intro函数的时候, intro函数的[[scope]]为:
“`
[[scope chain]] = [
{
name : 'laruence',
intor : undefined
}, {
window call object
}
]
“`

问题：
在这里为什么会有一个 `intor:undefined`?
不应该是
“`
[[scope chain]] = [
{
name : 'laruence',
}, {
window call object
}
]
“`
嘛？

深入理解**Javascript**之**this**关键字 | 一世浮华一场空 **(http://blog.vspsa.com/?p=15235)** | 15 Feb 2015 08:56

[...] 如我之前的文章所述(Javascript作用域), 定义在全局的函数, 函数的所有者就是当前页面, 也就是window对象. [...]

深入理解**Javascript**之**this**关键字 • 简简单单 **(http://fengjxblog.sinaapp.com/?p=171)** | 04 Dec 2014 00:05

[...] 如我之前的文章所述(Javascript作用域), 定义在全局的函数, 函数的所有者就是当前页面, 也就是window对象. [...]

Pages: **[2]** **1 (http://www.laruence.com/?p=863&cp=1#comments) » (http://www.laruence.com/?p=863&cp=1#comments) Show All (http://www.laruence.com/?p=863&cp=all#comments)**

Leave a Reply

Name

Mail (will not be published)

Website

(#)

CAPTCHA Code *

Submit Comment

Notify me of followup comments via e-mail

加关注 8.3万

Laruence

PHP (http://www.php.net/) 开发组成员, **Zend (http://www.zend.com/)** 兼职顾问, PHP7核心开发者, **Yaf (http://pecl.php.net/yaf)** , **Yar (http://pecl.php.net/yar)** , **Yac (http://pecl.php.net/Yac)** 等项目作者.

OpenSource Projects

Yaf (http://pecl.php.net/package/yaf) : PHP Framework in PHP extension
Yar (http://pecl.php.net/package/yar) : Light, concurrent RPC framework
Yac (http://pecl.php.net/yac) : PHP Contents cache
Yaconf (https://github.com/laruence/yaconf) : PHP Configurations Container
Taint (http://pecl.php.net/package/taint) : XSS code sniffer
Lua (http://pecl.php.net/package/lua) : Embedded lua interpreter
MsgPack (http://pecl.php.net/package/msgpack) :MessagePack in PHP extension
Couchbase (http://pecl.php.net/package/couchbase) : Libcouchbase wrapper
See also: laruence@github (http://github.com/laruence)

Advanced Random Posts

- Javascript原型链和原型的一个误区 (http://www.laruence.com/2010/05/13/1462.html)
- 关于Javascript的俩个有趣的探讨 (http://www.laruence.com/2009/09/27/1123.html)
- Serialize/Unserialize破坏单例 (http://www.laruence.com/2011/03/18/1909.html)
- 我对PHP5.4的一个改进 (http://www.laruence.com/2011/09/23/2171.html)
- PHP浮点数的一个常见问题的解答 (http://www.laruence.com/2013/03/26/2884.html)

Recent Comments

- 奶牛先生 on Yar – 并行的RPC框架(Concurrent RPC framework) (http://www.laruence.com/2012/09/15/2779.html/comment-page-1#comment-205700)
- Anonymous on PHP的版本发布历程 (http://www.laruence.com/2011/09/19/2148.html/comment-page-3#comment-205699)
- PHP程序员雷雪松 (http://www.leixuesong.cn) on 让PHP7达到最高性能的几个Tips (http://www.laruence.com/2015/12/04/3086.html/comment-page-1#comment-205698)
- Bleakwind on 上传进度支持(Upload progress in sessions) (http://www.laruence.com/2011/10/10/2217.html/comment-page-1#comment-205672)
- 刺客 (http://http://smismile.cnblogs.com/) on Yac (Yet Another Cache) – 无锁共享内存Cache (http://www.laruence.com/2013/03/18/2846.html/comment-page-2#comment-205668)
- Jaylen (http://mblwbhdxn.com) on Dom事件的srcTarget,strElement探幽 (http://www.laruence.com/2008/07/18/124.html/comment-page-1#comment-205613)
- Krystal (http://niujdqtfl.com) on 让你的PHP7更快(GCC PGO) (http://www.laruence.com/2015/06/19/3063.html/comment-page-1#comment-205612)
- Latasha (http://svcneu.com) on 深入理解PHP原理之Opcodes (http://www.laruence.com/2008/06/18/221.html/comment-page-2#comment-205610)
- Brandi (http://pcojmg.com) on Yaf 2.1性能测试(Yaf 2.1 Benchmark) (http://www.laruence.com/2011/12/02/2333.html/comment-page-1#comment-205609)
- Tish (http://uajeli.com) on 深入理解PHP原理之错误抑制与内嵌HTML (http://www.laruence.com/2009/07/27/1020.html/comment-page-1#comment-205608)

Tags

friends

- 80Sec (http://www.80sec.com/)
- cc0cc (http://www.54chen.com)
- CFC4N (http://www.cnxct.com)
- Demon (http://demon.at/)
- Errorrik (http://hi.baidu.com/erik168)
- glemir's (http://glemir.xplore.cn)
- Iterse's BLOG (http://blog.iterse.com/)
- Jessica (http://www.skiyo.cn/)
- moxie (http://blog.zoeey.org/)
- Pangee (http://pangee.me/)
- pplxh (http://pplxh.cublog.cn)
- rainX (http://www.rainx.cn/)
- Sara Golemon (http://blog.libssh2.org/)
- siko (http://www.chenbin.net/)
- stauren (http://stauren.net)
- Think in code (http://www.blankyao.cn)
- 三江小渡 (http://www.pureisle.net)
- 冰的河 (http://icyriver.net)
- 刘青炎 (http://liuqingyan.blogspot.com/)

- 孙清林博客 (http://www.sunqinglin.cn)
- 思考的Pyt (http://keyvalue.net/)
- 抚琴居 (http://www.yanbin.org/)
- 王洛革 (http://www.wangxuntian.com/)
- 神仙 (http://xiezhenye.com/)
- 黑夜路人 (http://blog.csdn.net/heiyeshuwu)

Visitor ClustrMaps

(http://www4.clustrmaps.com/counter/maps.php?url=http://www.laruence.com)