深入了解JavaScript，优化作用域链（2）

地址：http://blog.xieliqun.com/2016/10/06/scope-chain-2/

> 作为一个良好的开发者必需考虑程序的运行性能，作用域链的层级关系是JavaScript性能优化的一个重要部分。因为这关系到变量在内存里的读写速度。

尽量使用局部变量

从作用域链的结构可以看出，在执行上下文的作用域链中，标识符所在的位置越深，读写速度就会越慢。

全局变量总是存在于执行上下文作用域链的最末端，因此在标识符解析的时候，查找全局变量是最慢的，并且全局变量将常驻内存直到程序退出，而局部变量会在函数运行完直接销毁。所以，在编写代码的时候应尽量少使用全局变量，尽可能使用局部变量。

一个好的经验法则是：如果一个跨作用域的对象被引用了一次以上，则先把它存储到局部变量里再使用。例如：

```
function toggle(){
    if(document.getElementById('btn').className == 'active'){
        document.getElementById('btn').className = '';
        //do someThing
    }else{
        document.getElementById('btn').className = 'active';
        //do someThing
    }
}
```

以上代码中`document.getElementById('btn').className`被引用了三次，查找该变量必须遍历整个作用域链，直到最后在全局对象中才能找到`document`，然后再去找它的方法和属性，这样严重的影响了程序性能。我们可以改为：

```
function toggle(){
    var btnClass = document.getElementById('btn').className;
    if(btnClass == 'active'){
        btnClass = '';
        //do someThing
    }else{
        btnClass = 'active';
        //do someThing
    }
}
```

尽量不要改变作用域链

函数每次执行时对应的执行上下文都是独一无二的，所以多次调用同一个函数就会导致创建多个执行上下文，当函数执行完毕，执行上下文会被销毁。每一个执行上下文都和一个作用域链关联。一般情况下，在执行上下文运行的过程中，其作用域链只会被 with 语句和 catch 语句影响。

with语句是对象的快捷应用方式，用来避免书写重复代码。例如：

```
var o = {href:"github.com"};
var href = "blog.xieliqun.com";

function buildUrl(){
     var q = "?name=tsrot";
     with(o){
          var url = href+q;
     }
     return url;
}

var result = buildUrl();
alert(result);  //github.com?name=tsrot
alert(href);    //blog.xieliqun.com
```

第一个alert使用的o对象里的href，所以弹出github.com?name=tsrot，第二个alert使用的就是全局的href。

当代码运行到with语句时，运行期上下文的作用域链临时被改变了。一个新的可变对象被创建，它包含了参数指定的对象的所有属性。这个对象将被推入作用域链的头部，这意味着函数的所有局部变量现在处于第二个作用域链对象中，因此访问代价更高了。此时的作用域链为：

```
scope -> with.AO -> with.VO -> buildUrl.AO -> Global Object

with.AO = {
    url : undefined
}

with.VO = {
    href : "github.com"
}

buildUrl.AO = {
    q : "?name=tsrot"
}

Global Object = {
    o : {href:"github.com"},
    href : "blog.xieliqun.com",
    buildUrl : <reference to function>,
    result : undefined
}
```

另外一个会改变作用域链的是try-catch语句中的catch语句。当try代码块中发生错误时，执行过程会跳转到catch语句，然后把异常对象推入一个可变对象并置于作用域的头部。在catch代码块内部，函数的所有局部变量将会被放在第二个作用域链对象中。

闭包问题

一个函数只要内部函数未销毁（内部函数存在被调用的可能），这个函数就不会销毁，将一直存在于内存中，只有所有内部函数都销毁了，并所有的业务代码都已执行完，这个函数才会被释放。我们看看最常见的闭包问题：

```
function show(){
    var li = document.getElementsByTagName('li');
    var length = li.length;
    for(var i=0;i<length;i++){
        li[i].onclick = function(){
            alert(i);
        }
    }
}

show();
```

当点击li标签时，弹出的一直都是length的大小。这是一个比较经典的错误。为什么会这样呢？

由于show的内部函数（click事件处理程序时刻有调用可能），所以show的作用域链不能被销毁（只能页面卸载是销毁），i的值一直保持for循环执行完后的length值，此时的click的函数只是进行了声明而未运行，当click触发的时候，函数才开始执行，这个时候i的值已经是length了。所以每次触发onclick的时候才会alert length。我们进行改一下：

```
function show(){
    var li = document.getElementsByTagName('li');
    var length = li.length;
    for(var i=0;i<length;i++){
        (function(n){
            li[n].onclick = function(){
                alert(n);
            }
        })(i)
    }
}

show();
```

为什么这样就行了呢，这时候onclick引用的变量变成了n，而由于立即执行函数的原因，每个onclick函数在作用域链中分别保持着对应的n（0~length-1），这时候就可以了。

闭包会使子函数保持其作用域链的所有变量及函数与内存中，内存消耗很大，在使用的时候尽量销毁父函数不再使用的变量。你经常访问一些范围之外的标识符，每次访问都将导致一些性能损失。