链滴

理解 Javascript 中的作用域与作用域链

作者：zjhch123
原文链接：https://ld246.com/article/1493908807976
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

理解Javascript中的作用域与作用域链

最近恶补基础知识。非常感谢一像素博主的文章JS核心系列：浅谈函数的作用域，以及JavaScript 开进阶：理解 JavaScript 作用域和作用域链。这两篇文章对我的帮助非常大。

任何程序设计语言都有作用域，简单的说，作用域就是变量或函数的可访问范围。作用域控制着变量函数的可访问性与生命周期。

在绝大部分程序设计语言中，作用域的范围有全局作用域、函数作用域、块级作用域。

- 全局作用域：属于全局作用域的代码在任何地方都能访问到；
- 函数作用域：在定义该变量/函数的函数体内可以访问到；
- 块级作用域：在定义该变量/函数的语句块中可以访问到。

然而对于Javascript(ES5)来说，不存在块级作用域。

例如以下语句块：

```
if(true) {
  var foo = "bar";
}
console.log(foo); // bar

while(1) {
  var fooo = "baar";
  break;
}
console.log(fooo); // baar
```

当然，在es6中可以使用let关键字定义变量，使变量拥有块级作用域

```
while(1) {
  let foooo = "baaar";
  break;
}
console.log(foooo); // Uncaught ReferenceError: foooo is not defined;
```

js的函数作用域，具体可以体现在下例

```
function func1() {
  var x = 1;
  console.log(x);
}
func1();
console.log(x); // Uncaught ReferenceError: x is not defined;
```

在作用域内的变量没有被var声明，则自动提升为全局作用域。

```
function func2() {
  y = 1;
  console.log(y);
}
func2();
console.log(y); // 1
```

原文链接：理解 Javascript 中的作用域与作用域链

因为js存在“变量提升”的特性，所以在未使用严格模式('use strict')的情况下，在某一作用域内定义变量可以在任何位置被调用而不出错，哪怕是在其被声明之前。

```
console.log(upper);  // undefined
var upper = 3;
```

当某对象被调用时，js会依此按照

当前作用域 -> 次级作用域 -> 次次级作用域 -> ... -> 全局作用域

来查找并使用变量。此即为作用域链。

```
a = 1;
function add() {
  var b = 3;
  function in1() {
    var a = 2;
    console.log(a + b); // 5
  }
  function in2() {
    console.log(a + b); // 4
  }
  in1();
  in2();
}
add();
```

在in1()中，作用域链为 in1 -> add -> window，所以a + b此时的a为in1内定义的a，值为2；

在in2()中，作用域链为 in2 -> add -> window，所以a + b此时的a为window内定义的a，值为1。

在未使用严格模式时，可以使用with关键字临时扩展作用域。

```
a = 1;
function add() {
  var b = 3;
  function in1() {
    var a = 2;
    console.log(a + b); // 5
  }
  function in2(obj) {
    with(obj) {
      console.log(a + b); // 12
    }
  }
  var obj = {a: 9};
  in1();
  in2(obj);
}
add();
```

在in2()中，因为with关键字的原因，with内的对象被添加到作用域链的头部，形成了如下作用域链：

obj -> in2 -> add -> window

因此，此时in2内with语句块内的a值为9。

原文链接：理解 Javascript 中的作用域与作用域链

以上，可以看出，在运行期上下文中，访问全局变量的速度是最慢的。因为全局作用域总是处在作用链的最末端。

所以，在编写代码的时候，尽可能少使用全局变量，多使用局部变量。

有前人总结了经验，是：如果有一个变量跨作用域引用了一次以上，则先将它存储到局部变量里再使。

在面试中经常讲this和作用域混合在一起出题。所以这里也说说this。

在一个函数中(es5)，this总是指向调用该函数的对象，总是在运行时才能确定this的值以及其指向。

```
var obj = {
  x: 1,
  log: function() {
    console.log(this.x);
  }
};

obj.log(); // 1
var func3 = obj.log;
func3(); // undefined
```

直接调用函数的话，函数体内的this默认指向window。

```
window.name = "func4";
function func4() {
  console.log(this.name);
}
func4(); // func4;
```

使用call, bind, apply可以动态改变函数体内this的指向。使用new关键字调用函数也会使函数体内的this指向发生变化。这些留给以后说。

在一像素的博客内有两道题，我这边照搬过来。这两道题是对作用域、this最好的锻炼题。

```
// code1
var foo = "window";
var obj = {
    foo : "obj",
    getFoo : function(){
       return function(){
            return this.foo;
        };
    }
};
var f = obj.getFoo();
f(); // window

 // code2
var foo = "window";
var obj = {
    foo : "obj",
    getFoo : function(){
       var that = this;
        return function(){
```

原文链接：理解 Javascript 中的作用域与作用域链

```
            return that.foo;
        };
    }
};
var f = obj.getFoo();
f(); // obj
```

以上所有测试代码，都可以在我的github内下载并调试。

原文链接：理解 Javascript 中的作用域与作用域链