高级Day 04---作用域链和闭包

一、匿名函数

1.1 匿名函数的概念

声明一个没有函数名的函数，就是匿名函数。

有函数名的函数就是具名函数。

> 看下面的代码：

```
<script type="text/javascript">
    /*
    //这里定义了一个函数，而且没有函数名。这样写语法是错误的,如果允许这样定义，那么根本就没有办法调用。
    //所以，我们可以用一个变量来存储一下
    function(){

    }
    */
  // 声明了一个匿名函数，并把匿名函数赋值给变量f。 注意这个时候这个匿名函数并没有执行。
  var f = function(){
    alert("哥们我是匿名函数内的代码");
  }
  //我们可以把变量 f 当做一个函数名来调用
  f();  //调用上面定义的匿名函数
</script>
```

1.2 匿名函数的应用场景

有些场景大家已经比较熟悉了。

1.2.1 给标签绑定事件

```
<script type="text/javascript">
    var btn = document.getElementById("btn");
    btn.onclick = function () {
        alert("点我干吗");
    }
</script>
```

1.2.2 在定时器中使用

```
<body>
    <h1></h1>
    <script type="text/javascript">
        var showTimeArar = document.getElementsByTagName("h1")[0];
        setInterval(function () {
            showTimeArar.innerHTML = new Date().toLocaleString();
        }, 1000);
    </script>
</body>
```

1.2.3 给对象定义方法

```
<script type="text/javascript">
    var person = {
        name : "凤姐",
        age : 30,
        play : function () {
            alert(this.name + "在美国玩");
        }
    }
    person.play();
</script>
```

1.3 匿名函数的自调用

> 有些场景，我们需要定义完函数之后立即执行，这个时候可以定义一个匿名函数来完成。

```
(function () {
    alert("匿名函数立即执行")
})();
```

> 说明

1. 需要把匿名函数用一对圆括号括起来，把匿名函数作为一个整体来对待
2. 最后再添加一对圆括号表示调用函数。这样定义的匿名函数就会立即执行
3. 当然，这个时候即使给这个函数加上方法名，也可以调用。不过这种情况为什么还要加方法名呢?

二、变量的作用域

> 变量的作用域指的是，变量起作用的范围。也就是能访问到变量的有效范围。
>
> JavaScript的变量依据作用域的范围可以分为：
>
> - 全局变量
> - 局部变量

2.1 全局变量

> 定义在函数外部的变量都是全局变量。

> 全局变量的作用域是当前文档，也就是当前文档所有的JavaScript脚本都可以访问到这个变量。
>
> 下面的代码是书写在同一个HTML文档中的2个JavaScript脚本:

```
<script type="text/javascript">
    //定义了一个全局变量。那么这个变量在当前html页面的任何的JS脚本部分都可以访问到。
    var v = 20;
    alert(v); //弹出：20
</script>
<script type="text/javascript">
    //因为v是全局变量，所以这里仍然可以访问到。
    alert(v);  //弹出：20
</script>
```

> 再看下面一段代码：

```
<script type="text/javascript">
    alert(a);
    var a = 20;
</script>
```

> 运行这段代码并不会报错， alert(a); 这行代码弹出：undefined。

为什么在声明 a 之前可以访问变量 a 呢? 能访问 a 为什么输出是undefined而不是20呢？

声明提前！

- 所有的全局变量的声明都会提前到JavaScript的前端声明。也就是所有的全局变量都是先声明的，并且早于其他一切代码。
- 但是变量的赋值的位置并不会变，仍然在原位置赋值。

> 所以上面的代码等效下面的代码：

```
<script type="text/javascript">
    var a; //声明提前
    alert(a);
    a = 20; //赋值仍然在原来的位置
</script>
```

2.2 局部变量

> 在函数内声明的变量，叫局部变量！表示形参的变量也是局部变量！
>
> 局部变量的作用域是局部变量所在的整个函数的内部。 在函数的外部不能访问局部变量。

```
<script type="text/javascript">
    function f(){
        alert(v);  //    弹出：undefined
        var v = "abc";  // 声明局部变量。局部变量也会声明提前到函数的最顶端。
        alert(v);   //  弹出：abc
    }
    alert(v);  //报错。因为变量v没有定义。 方法 f 的外部是不能访问方法内部的局部变量 v 的。
 </script>
```

2.3 全局变量和局部变量的一些细节

> 看下面一段代码:

```
<script type="text/javascript">
    var m = 10;
    function f(){
        var m = 20;
        alert("方法内部：" + m);  //代码1
    }
    f();
    alert("方法外部：" + m); //代码2
</script>
```

在方法内部访问m，访问到的是哪个m呢？局部变量的m还是全局变量的m？

2.3.1 全局变量和局部变量重名问题

1. 在上面的代码中，当局部变量与全局变量重名时，局部变量的作用域会覆盖全局变量的作用域。也就是说在函数内部访问重名变量时，访问的是局部变量。所以 "代码1" 部分输出的是20。
2. 当函数返回离开局部变量的作用域后，又回到全局变量的作用域。所以代码2输出10。
3. 如何在函数访问同名的全局变量呢？通过：window.全局变量名

```
<script type="text/javascript">
    var m = 10;
    function f(){
        var m = 20;
        alert(window.m);  //访问同名的全局变量。其实这个时候相当于在访问window这个对象的属性。
    }
    f();
</script>
```

2.3.2 JavaScript中有没有块级作用域？

> 看下面一段代码:

```
<script type="text/javascript">
  var m = 5;
  if(m == 5){
    var n = 10;
  }
  alert(n); //代码1
</script>
```

> 代码1输出什么？ undefined还是10？还是报错？
>
> 输出10！

- JavaScript的作用域是按照函数来划分的
- JavaScript没有块级作用域

在上面的代码中，变量 n 虽然是在 if 语句内声明的，但是它仍然是全局变量，而不是局部变量。

只有定义在方法内部的变量才是局部变量

> 注意：

- 即使我们把变量的声明放在 if、for等块级语句内，也会进行声明提前的操作！

三、作用域链---作用域的深入理解

3.1 执行环境

执行环境（ execution context ）是 JavaScript 中最为重要的一个概念。执行环境定义了变量或函数有权访问的其他数据，决定了它们各自的行为。每个执行环境都有一个与之关联的 变量对象（variable object），环境中定义的所有变量和函数都保存在这个对象中。虽然我们编写的代码无法访问这个对象，但解析器在处理数据时会在后台使用它。

全局执行环境是最外围的一个执行环境。在 Web 浏览器中，全局执行环境被认为是 window 对象，因此所有全局变量和函数都是作为 window 对象的属性和方法创建的。对全局执行环境变量来说，变量对象就是window对象，对函数来说，变量对象就是这个函数的 活动对象 ，活动对象是在函数调用时创建的一个内部变量。

每个函数都有自己的执行环境，当执行流进入一个函数时，函数的执行环境就会被推入一个执行环境栈中。而在函数执行之后，栈将执行结束的函数的执行环境弹出，把控制权返回给之前的执行环境。

3.2 作用域链

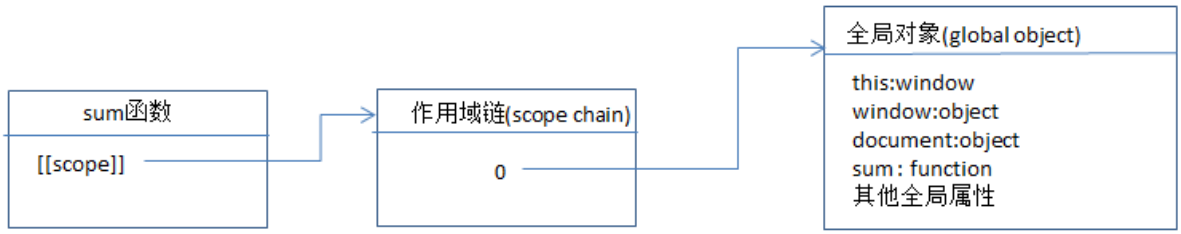
作用域链与一个执行环境相关，作用域链用于在标示符解析中变量查找。

在JavaScript中，函数也是对象，实际上，JavaScript里一切都是对象。函数对象和其它对象一样，拥有可以通过代码访问的属性和一系列仅供JavaScript引擎访问的内部属性。其中一个内部属性是[[Scope]]，由ECMA-262标准第三版定义，他就指向了这个函数的作用域链。作用域链中存储的是与每个执行环境相关 变量对象 (函数内部也是活动对象)。

当创建一个函数(声明一个函数)后，那么会创建这个函数的作用域链。这个函数的作用域链在这个时候只包含一个变量对象(window)

```
<script type="text/javascript">
    function sum(num1, num2){
        var sum = num1 + num2;
        return sum;
    }
</script>
```

> 函数 sum 的作用域链示意图：



> 说明：

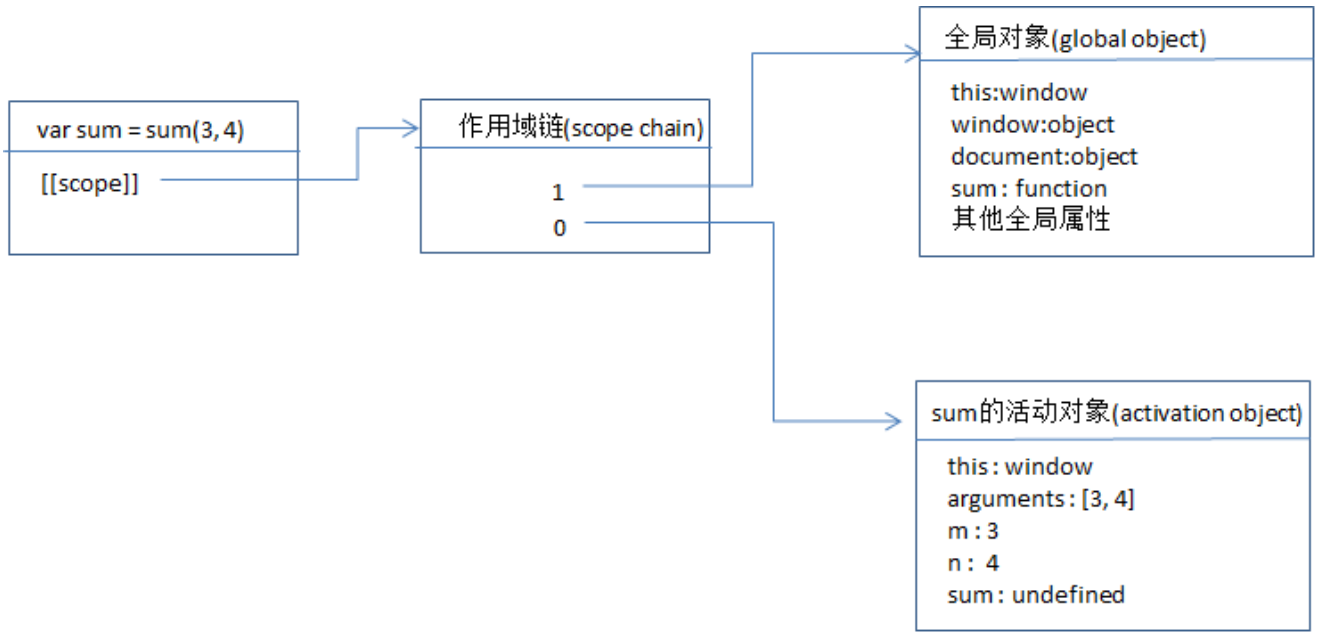
- 函数创建的时候，这个时候作用域链中只有一个 变量对象 (window)

> 当执行下面的代码：

```
<script type="text/javascript">
    function sum(num1, num2){
        var sum = num1 + num2;
        return sum;
    }
    var sum = sum(3, 4);
</script>
```

当调用 sum 函数时，会首先创建一个 “执行环境”，这个 执行环境 有自己的作用域链，这个作用域链初始化为 sum 函数的 [[scope]] 所包含的对象。然后创建一个 与这个执行环境相关的 变量对象(活动对象)，这个 变量对象 中存储了在这个函数中定义的所有参数、变量和函数。把 变量对象 存储在作用域中的顶端。 以后在查找变量的时候，总是从作用域链条的顶端开始查找，一直到作用域链条的末端。

> 看下面的示意图：



> 说明：

1. 在sum中访问一个变量的时候，总是从作用域链的顶端开始查找，如果找到就得到结果，如果找到不到就一直查找，直到作用域链的末端。
2. 因为在方法内的存在变量和函数的声明提前现象，所以函数一旦执行 函数的活动对象(变量对象)中总是保存了这个韩硕中声明的所有变量和函数。
3. 如果在函数中又定义了一个内部函数(还没有执行)，则这个时候内部函数的作用域，是包含了外部函数的作用域。 一旦内部函数开始执行则把自己的活动对象添加到了这个作用域的顶端。

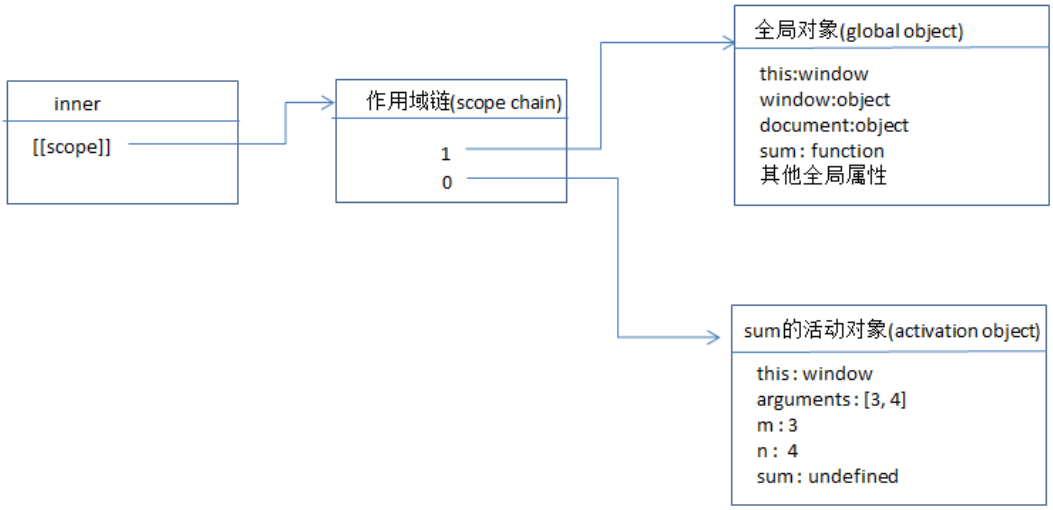
```
<script type="text/javascript">
    function sum(num1, num2){
        var sum = num1 + num2;
        function inner (a) {

        }

        return sum;
    }

    var sum = sum(3, 4);
</script>
```

> 内部函数的作用域：



> 函数执行后的作用域示意图不再画出。

四、闭包

> 看下面的代码：

```
<script type="text/javascript">
    function createSumFunction(num1, num2){
        return function () {
            return num1 + num2;
        };
    }

    var sumFun = createSumFunction(3, 4);
    var sum = sumFun();
    alert(sum);
</script>
```

在上面的代码中，createSumFunction函数返回了一个匿名函数，而这个匿名函数使用了createSumFunction函数中的局部变量(参数)，即使createSumFunction这个函数执行结束了，由于作用域链的存在，他的局部变量在匿名函数中仍然可以使用，这个匿名函数就是闭包。

闭包是指有权访问另一个函数作用域中的变量的函数。

五、闭包的应用

5.1 返回外部函数的局部变量

```
<script type="text/javascript">
    function outer () {
         var num = 5;
        //定义一个内部函数
         function inner () {
            //内部函数的返回值是外部函数的一个局部变量
             return num;
         }
        //把局部变量的值++
         num++;
        // 返回内部函数
         return inner;
    }
    var num = outer()();  // 6
    alert(num);
</script>
```

> 说明：

1. 这例子中，虽然函数的声明在num++之前，但是函数返回的时候num已经++过了，所以只是num自增之后的值。
2. 结论：闭包中使用的局部变量的值，一定是局部变量的组后的值。

5.2 使用函数自执行和闭包封装对象

> 封装一个能够增删改查的对象

```
<script type="text/javascript">
    var person = (function () {
        //声明一个对象，增删改查均是针对这个对象
        var personInfo = {
            name : "李四",
            age : 20
        };
        //返回一个对象，这个对象中封装了一些对personInfor操作的方法
        return {
            //根据给定的属性获取这个属性的值
            getInfo : function (property) {
                return personInfo[property];
            },
            //修改一个属性值
            modifyInfo : function (property, newValue) {
                personInfo[property] = newValue;

            },
            //添加新的属性
            addInfo : function (property, value) {
                personInfo[property] = value;

            },
            //删除指定的属性
            delInfo : function (property) {
                delete personInfo[property];

            }
        }
    })();
    alert(person.getInfo("name"));
    person.addInfo("sex", "男");
    alert(person.getInfo("sex"));
</script>
```

5.3 for循环典型问题

> 看下面的代码

```
<body>
    <input type="button" value="按钮1"     >
    <input type="button" value="按钮2"     >
    <input type="button" value="按钮3"     >
    <script type="text/javascript">
        var btns = document.getElementsByTagName("input");
        for (var i = 0; i < 3; i++) {
            btns[i].onclick = function () {
                alert("我是第" + (i + 1) + "个按钮");
            };
        }
    </script>
</body>
```

发现在点击三个按钮的时候都是弹出 我是第4个按钮。 为什么呢？闭包导致的！ 每循环一次都会有一个匿名函数设置点击事件，闭包总是保持的变量的最后一个值，所以点击的时候，总是读的是 i 的组后一个值4.

> 解决方案1：给每个按钮添加一个属性，来保存 每次 i 的临时值

```
<body>
    <input type="button" value="按钮1"     >
    <input type="button" value="按钮2"     >
    <input type="button" value="按钮3"     >
    <script type="text/javascript">
        var btns = document.getElementsByTagName("input");
        for (var i = 0; i < 3; i++) {
            //把i的值绑定到按钮的一个属性上，那么以后i的值就和index的值没有关系了。
            btns[i].index = i;
            btns[i].onclick = function () {
                alert("我是第" + (this.index + 1) + "个按钮");
            };
        }
    </script>
</body>
```

> 解决方案2：使用匿名函数的自执行

```
<body>
    <input type="button" value="按钮1"    >
    <input type="button" value="按钮2"    >
    <input type="button" value="按钮3"    >
    <script type="text/javascript">
        var btns = document.getElementsByTagName("input");
        for (var i = 0; i < 3; i++) {
            //因为匿名函数已经执行了，所以会把 i 的值传入到num中，注意是i的值，所以num
            (function (num) {
                btns[i].onclick = function () {
                    alert("我是第" + (num + 1) + "个按钮");
                }
            })(i);
        }
    </script>
</body>
```