

函数（下）

0.1. 主要内容：



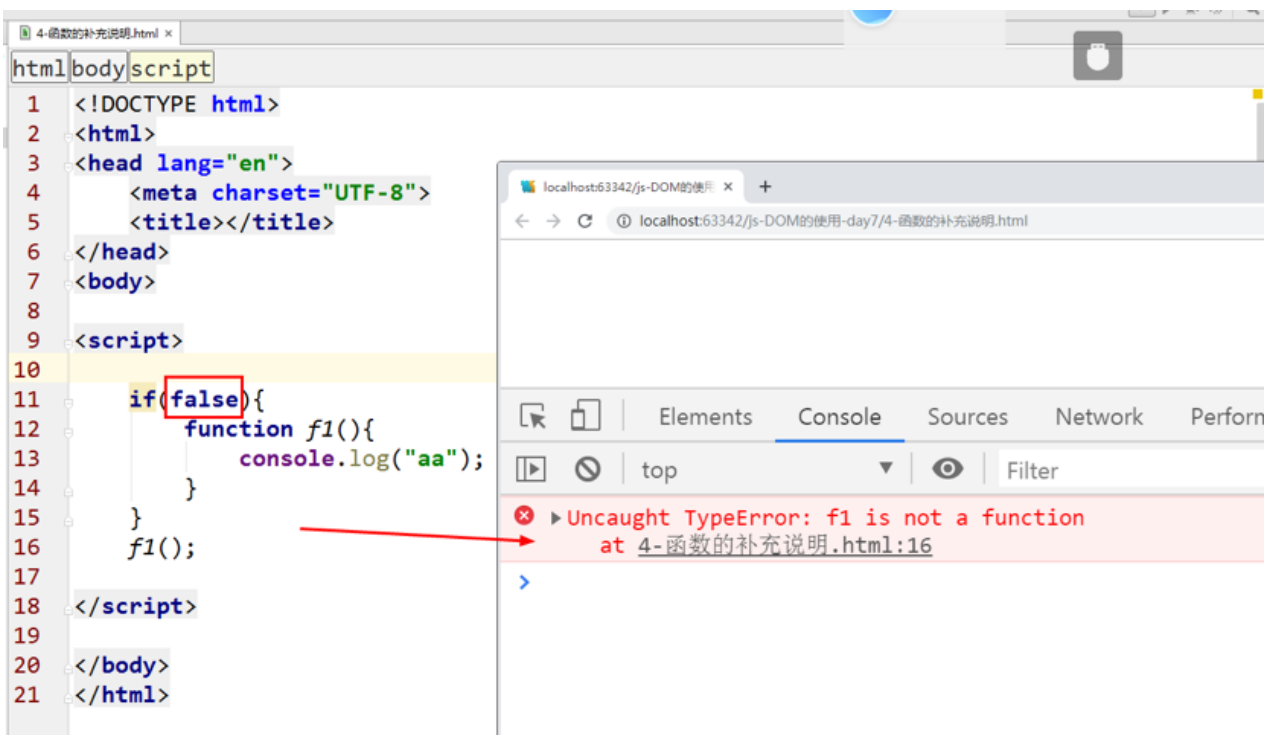
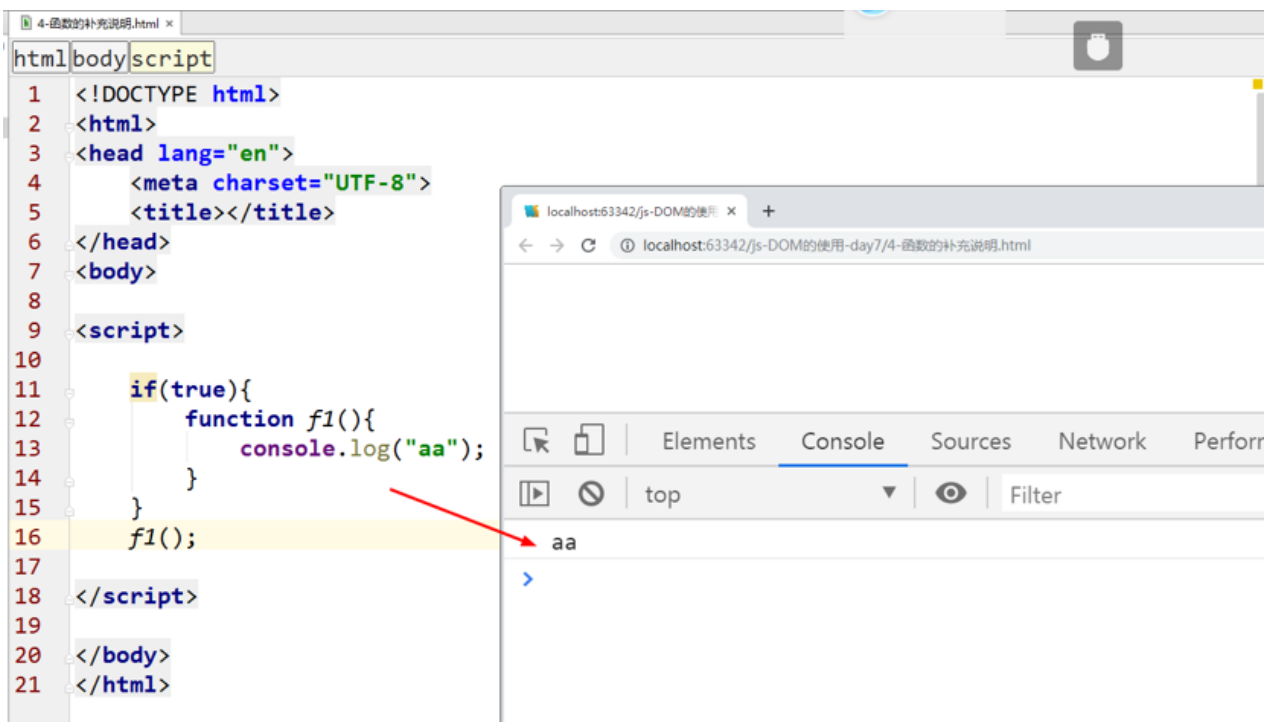
0.1. 学习目标：

节数	知识点	要求
第一节（函数总结说明）	函数总结说明	掌握
第二节（闭包）	闭包	掌握

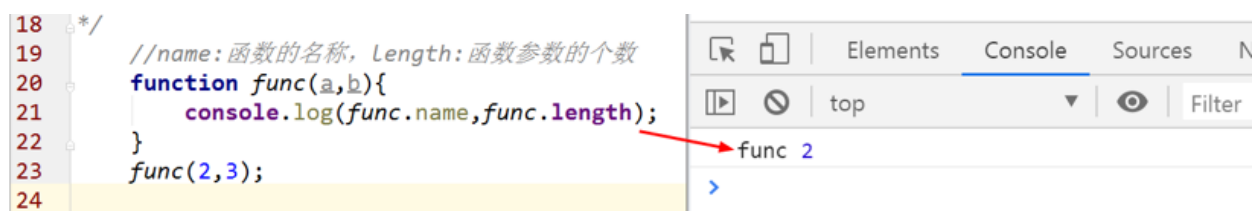
1. 函数总结说明

1.1. 不能在非函数的代码块中声明函数

不要在代码块中定义一个函数，这样造成假的情况下不执行

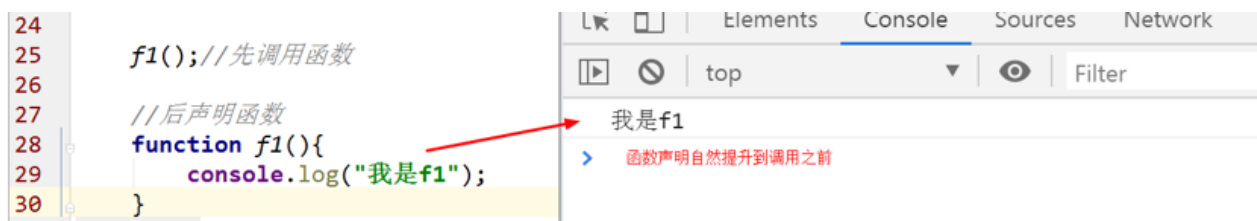


1.2. name属性和length属性



1.3. 变量和函数的提升

函数的提升



1.4. 超一等公民

在很多传统语言（C/C++/Java/C#等）中，函数都是作为一个二等公民存在，你只能用语言的关键字声明一个函数然后调用它，如果需要把函数作为参数传给另一个函数，或是赋值给一个本地变量，又或是作为返回值，就需要通过函数指针(function pointer)、代理(delegate)等特殊的方式周折一番。但是在JavaScript世界中函数却是超一等公民，它不仅拥有一切传统函数的使用方式（声明和调用），而且可以做到像简单值一样赋值、传参、返回，这样的函数也称之为第一级函数（First-class Function）或一等公民。不仅如此，JavaScript中的函数还充当了类的构造函数的作用，同时又是一个Function类的实例(instance)。这样的多重身份让JavaScript的函数变得非常重要。

1.5. 函数自定义声明

1.6. 将函数赋值给变量

1.7. 将函数作为参数

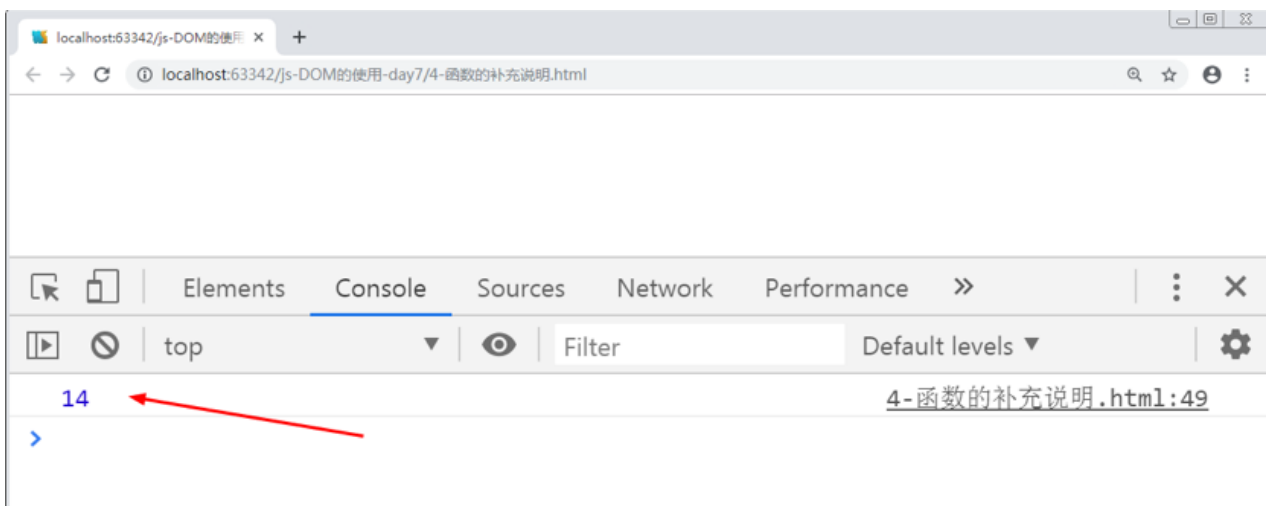
1.8. 将函数自调用

1.9. 将函数作为对象的方法

1.10. arguments对象

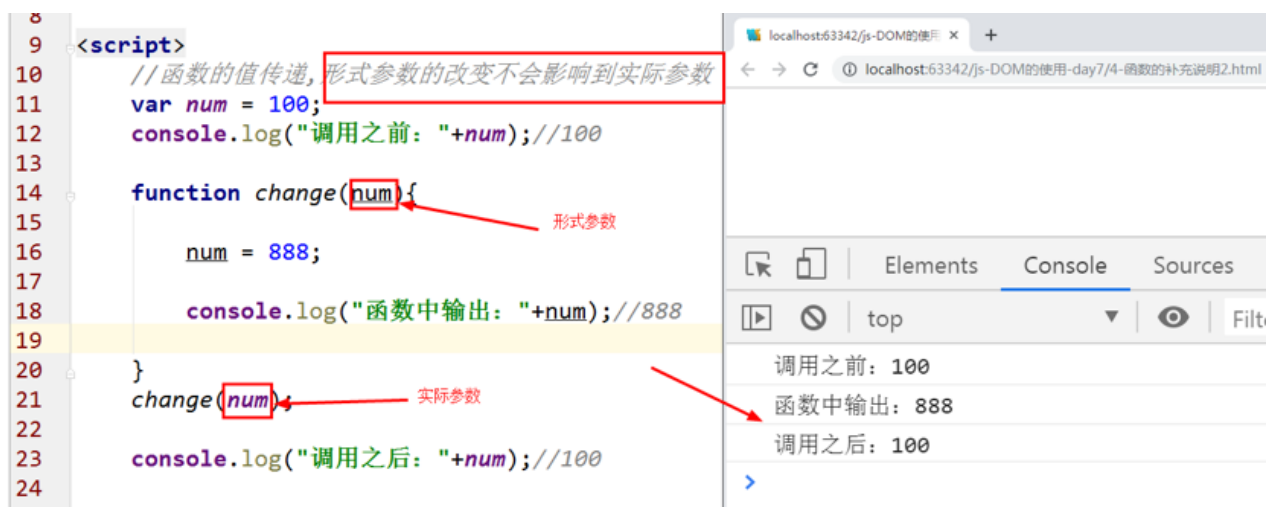
Arguments对象是js给我们提供的一个形式参数，它是一个数组。

```
43 }())*/
44 //arguments[0]:形式参数1, arguments[1]: 形式参数2, arguments[2]: 形式参数3
45 function sum(){
46     return arguments[0] + arguments[1] + arguments[2];
47     //return arguments;
48 }
49 console.log(sum(2,5,7));
50 // console.log(sum());
```



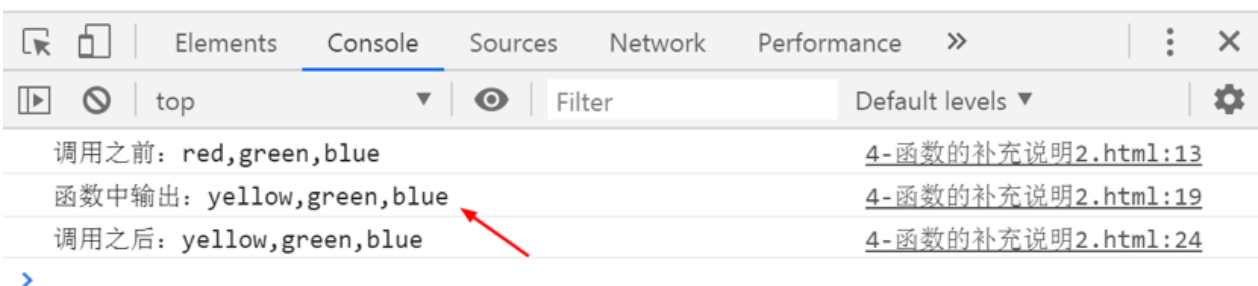
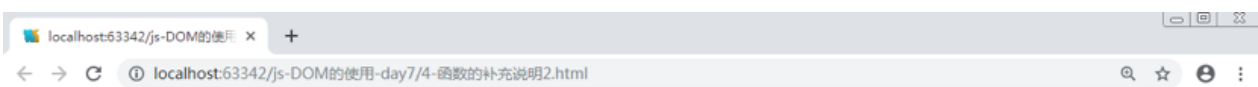
1.11. 值传递和地址传递

值传递：



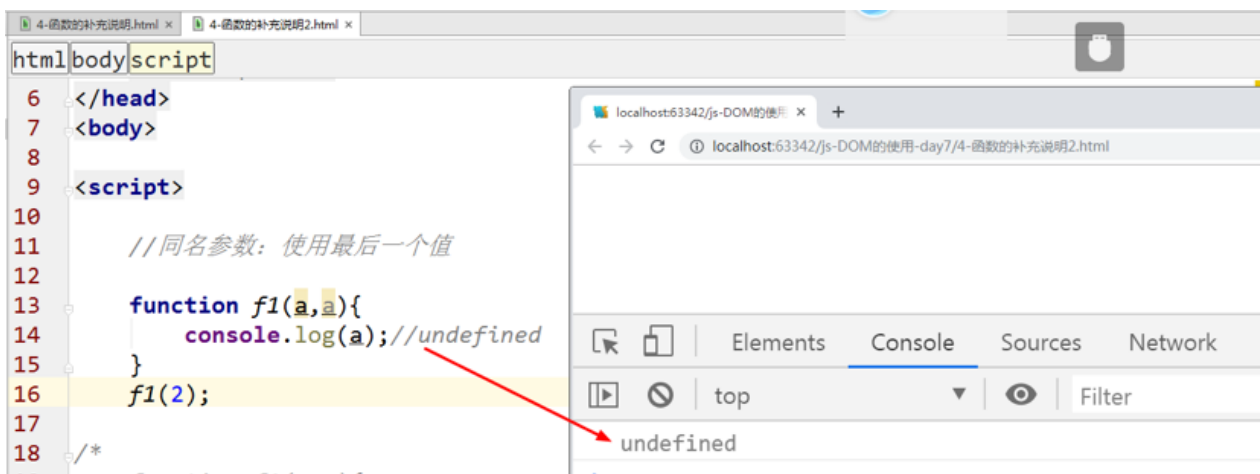
地址传递：

```
html body script
8
9 <script>
10 //地址传递: 实际参数和形式参数公用一个地址, 形式参数的改变会影响到实际参数
11 var arr = ["red", "green", "blue"];
12
13 console.log("调用之前: " + arr); // "red", "green", "blue"
14
15 function change(val){
16     val[0] = "yellow";
17     console.log("函数中输出: " + val); // "yellow", "green", "blue"
18 }
19 change(arr);
20
21 console.log("调用之后: " + arr); // "yellow", "green", "blue"
22
23
24
25
```



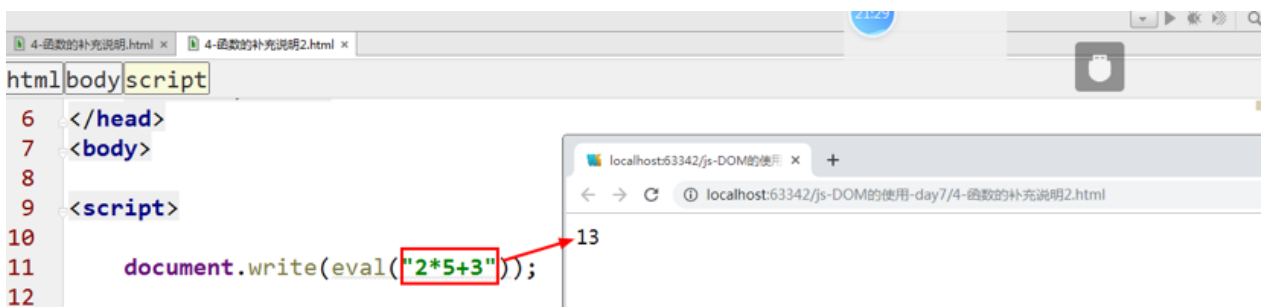
```
html body script
9 <script>
10
11 var arr = ["red", "green", "blue"];
12
13 console.log("调用之前: " + arr); // "red", "green", "blue"
14
15 function change(val){
16     val = ["brown", "grey", "yellow"]; // 开辟了一个新的地址
17     console.log("函数中输出: " + val); // "brown", "grey", "yellow"
18 }
19 change(arr);
20
21 console.log("调用之后: " + arr); // "red", "green", "blue"
22
23
24
25
```

1.12. 函数的同名参数



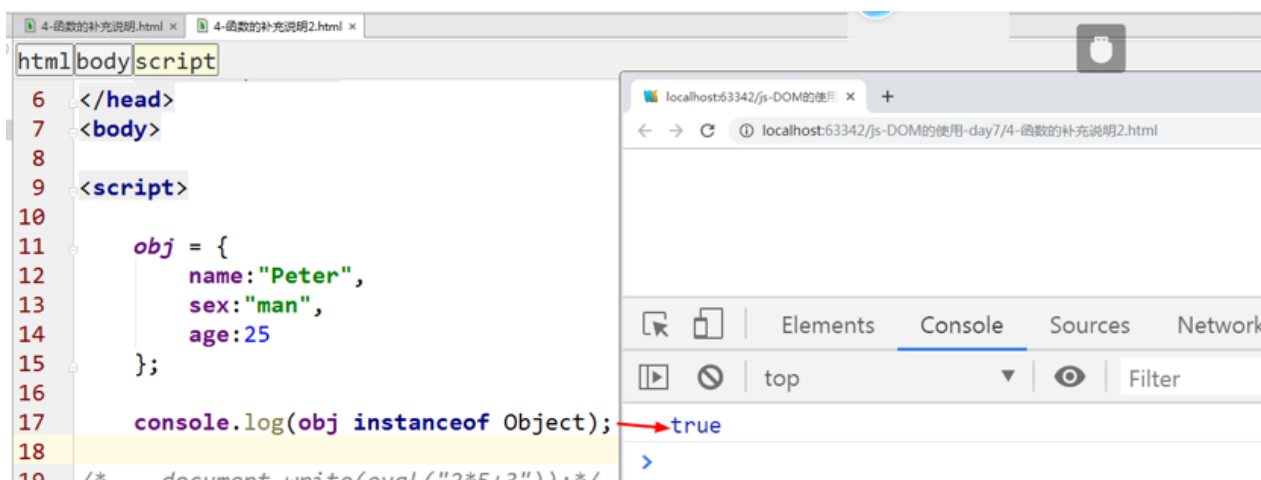
1.13. eval函数

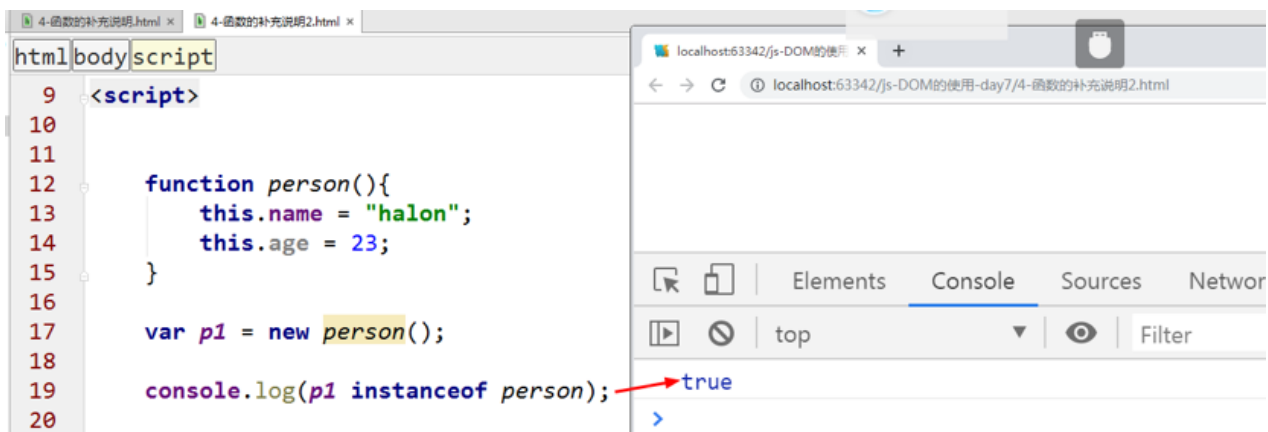
eval("字符串"):可以计算字符串的值



1.14. instanceof类型检测

instanceof 用于判断一个变量是否是某个对象的实例





1.15. javascript垃圾回收机制

对于其他语言来说，如C,C++,需要开发者手动的来跟踪并管理内存。

Javascript 具有自动垃圾回收机制，会定期对那些我们不再使用的变量、对象所占用的内存进行释放

其原理就是找出那些不在被使用的变量，然后释放其所占有的内存。回收器一般是按照固定的时间间隔或者预设的时间进行处理的。

1.16. 作业：统计字符串中字母出现最多的字符

2. 闭包（重点难点）

2.1. 为什么要使用闭包

闭包是解决什么问题的？

如何在函数外部访问函数内部局部变量的问题。

```
14-作用域和作用域的练习.html x 15-闭包.html x 02:57
html body script
4 <meta charset="UTF-8">
5 <title></title>
6 </head>
7 <body>
8
9 <script>
10 //在函数外部访问函数内部的局部变量
11 function f1(){
12
13     var a = 100
14
15 }
16
17 f1();
18
19 alert(a); //报错
20
21 </script>
22
23 </body>
24 </html>
```

怎样解决这个问题？

可以使用return语句返回一个值，但这个不是闭包。

```
20 <script>
21
22 function f2(){
23     var num = 99;
24     return num;
25 }
26 console.log(f2());
27
```

2.2. 什么是闭包

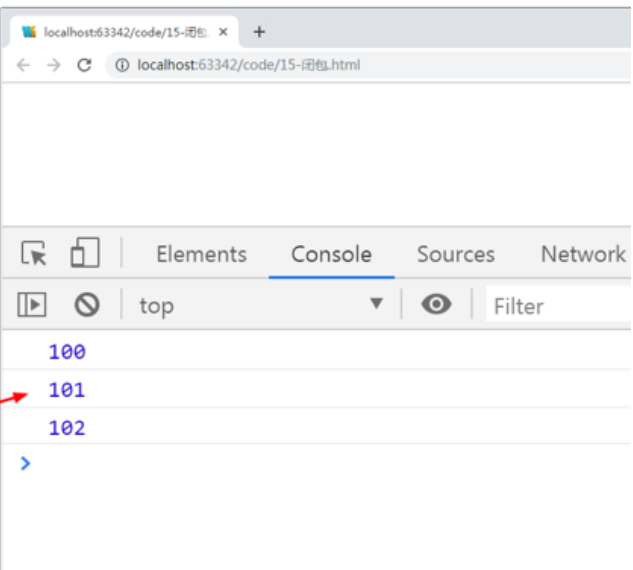
闭包：能够读取其他函数（外部函数）变量的函数（内部函数）就是闭包。

```
19
20 <script>
21
22 function f2(){
23     var num = 100;
24     function inner(){
25         console.log(num);
26     }
27     inner();
28 }
29 f2();
30
```

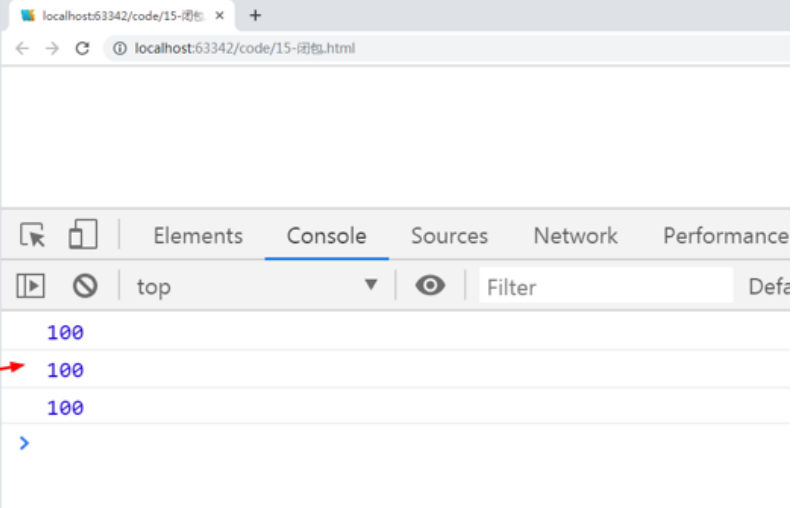
2.3. 闭包的分类

1、函数闭包

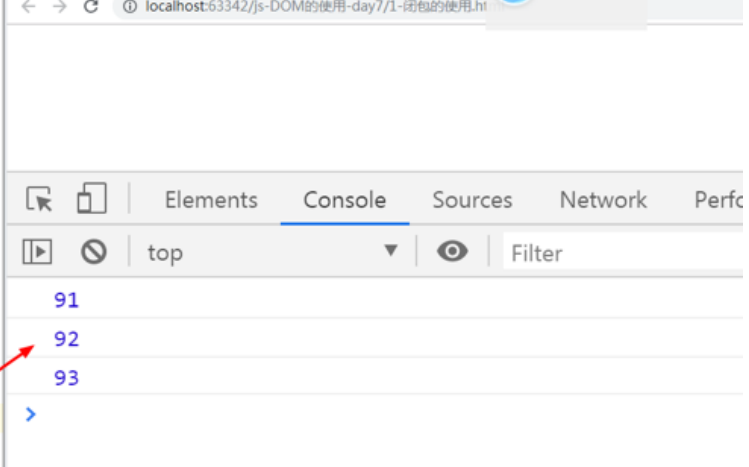
```
41 // 闭包: 能够访问其他函数(f2)变量的函数(f1) 就是闭包。
42 function f1(){
43     var n = 99;
44     function f2(){
45         n++;
46         return n;
47     }
48     return f2;
49 }
50
51
52 var re = f1();
53
54 // console.log(f1()); //f2这个函数-100
55
56 console.log(re()); //100
57 console.log(re()); //101
58 console.log(re()); //102
59 </script>
60
61 </body>
```



```
59
60 function f1(){
61     var n = 99;
62     function f2(){
63         n++;
64         console.log(n);
65     }
66     f2();
67 }
68
69
70
71 f1(); //100
72 f1(); //100
73 f1(); //100
74
75 </script>
76
```



```
45
46 function f1(){
47     var n = 90;
48     return function (){
49         n++;
50         return n;
51     }
52 }
53
54
55 var re = f1(); //f2()这个函数
56
57 console.log(re()); //91
58 console.log(re()); //92
59 console.log(re()); //93
60
```



```
15-闭包.html x 1-闭包的使用.html x
html body script
46 // 在函数外部不能访问函数内的局部变量
47 function f1(){
48     var n = 99;
49 }
50 f1();
51 console.log(n);
52
53 // 可以通过闭包的方式来访问函数内的局部变量
54 function f1(){
55     var n = 99; // 对于f2这个函数来说，n就是全局变量
56     // var m = 8;
57     function f2(){
58         n++;
59         return n;
60     }
61     return f2;
62 }
63
64 var re = f1(); // f2() 这个函数
65 // 我们调用了三次f2这个函数
66 console.log(re()); // 91
67 console.log(re()); // 92
68 console.log(re()); // 93
```

2、对象闭包

```
2-对象闭包的使用.html x
html body script
9 <script>
10 // 对象闭包
11 var name = "the window";
12 // 定义一个对象
13 var obj = {
14     name: "my object",
15     getName: function(){
16         return function(){
17             return this.name; // this指向window
18         }
19     }
20 }
21
22 // 我们调用obj.getName这个方法，我们将它赋值一个变量re，这个变量re在全局作用域，
23 // 那么re这个变量就属性window对象，所以这个this指向window对象
24 var re = obj.getName();
25
26
27 console.log(re()); // the window
```

2.4. 闭包的特点和作用

闭包的特点：

特点1：函数内部包含一个函数

特点2：内部函数可以使用外部函数的变量

闭包的作用：

- 1.一个就是可以读取函数内部的变量。
- 2.另一个就是让这些变量的值始终保持在内存中。

2.5. 计数器的累加

The screenshot shows a code editor with the following HTML and JavaScript code:

```
html body script
4     <meta charset="UTF-8">
5     <title></title>
6 </head>
7 <body>
8
9 <script>
10
11     // 计数器的累加
12     function jsq(){
13         var n = 7; // 局部变量不能实现计数器累加
14         n++;
15         console.log(n);
16     }
17     jsq();//8
18     jsq();//8
19     jsq();//8
20 </script>
21
22 </body>
23 </html>
```

The browser console shows the output of the function calls:

```
top
3 8
>
```

A red arrow points from the third call to `jsq()` in the code to the value `8` in the console, indicating that the counter did not increment.

The screenshot shows a code editor with the following HTML and JavaScript code:

```
21     var n = 7; // 全局变量可以实现计数器累加
22
23     function jsq(){
24
25         n++;
26
27         console.log(n);
28     }
29     jsq();//8
30     jsq();//9
31     jsq();//10
32 </script>
33
34 </body>
35 </html>
```

The browser console shows the output of the function calls:

```
top
8
9
10
>
```

A red arrow points from the third call to `jsq()` in the code to the value `10` in the console, indicating that the counter incremented correctly because it used a global variable.

2.6. 闭包的问题

内存泄漏

内存泄漏（Memory Leak）是指程序中已动态分配的堆内存由于某种原因程序未释放或无法释放，造成系统内存的浪费，导致程序运行速度减慢甚至系统崩溃等严重后果。

多次调用

2.7. 闭包的实际应用

没有使用闭包

```
35
36 for(let i = 0;i<oli.length;i++){
37 //    oli[i].index = i;
38     oli[i].onmouseover = function(){
39         oli[i].style.background = "yellow";
40     }
41
42     oli[i].onmouseout = function(){
43         oli[i].style.background = "";
44     }
45 }
46
47
```

使用var

使用了闭包

```
3-闭包的实际应用.html x
htmlbodyscript
<ul>
10 <li>第一行</li>
11 <li>第二行</li>
12 <li>第三行</li>
13 </ul>
14
15 <script>
16     var oli = document.querySelectorAll("li");//获取li元素-是一个数组
17
18     //这是闭包
19     for(var i = 0;i<oli.length;i++){
20
21         oli[i].onmouseover = showbgcolor(i,"blue");
22
23         oli[i].onmouseout = showbgcolor(i,"");
24
25     function showbgcolor(i,color){
26
27         return function (){
28             oli[i].style.background = color;
29         }
30     }
31 }
32
```

