# 第15天【对象和函数】

## 主要内容

1. 对象
2. 函数

## 学习目标

|  |  |  |
| --- | --- | --- |
| 节数 | 知识点 | 要求 |
| 第一节 | 对象的概述 | 了解 |
| 健名 | 掌握 |
| 键值 | 掌握 |
| 对象的创建 | 掌握 |
| 对象属性的读写 | 掌握 |
| 第二节 | 对象属性的操作 | 了解 |
| 对象的遍历 | 掌握 |
| 对象的引用 | 掌握 |
| 第三节 | 函数的概述 | 了解 |
| 函数的创建 | 掌握 |
| 函数的调用 | 掌握 |
| 函数的参数 | 掌握 |
| 第四节 | 函数的作用域 | 掌握 |

1. **对象**

**1.1对象的概述**

对象（object）是大括号定义的无序的数据集合，由键值对构成，键与值之间用冒号分隔，如果对象内部包含多个键值对，每个键值对之间用逗号分隔，最后一个键值对末尾不用加逗号。大括号末尾要使用分号表示对象定义结束。

对象是JavaScript的核心概念，也是最重要的数据类型。JavaScript的所有数据都可以被视为对象。此外，JavaScript 允许自定义对象。

|  |
| --- |
| var obj = { key : value }; |

上面代码定义了一个对象，它被赋值给变量obj。key是“键名”，value是“键值”。

**1.2健名**

键名也被称为属性(property)，对象的所有属性都是字符串，所以加不加引号都可以。但是，如果属性不符合标识符的条件(比如第一个字符为数字，或者含有空格或运算符)，则必须加上引号。

|  |
| --- |
| var obj = {  '1p': "Hello World",  'h w': "Hello World",  'p+q': "Hello World"  }; |

上面对象的三个属性，都不符合标识名的条件，所以必须加上引号。

**1.3键值**

键值是属性所对应的具体的值。javascript的对象的键值可以是任何数据类型。

|  |
| --- |
| var frank = {  name: "Frankenstein Add",  age: 18,  sex: "male",  ability: function eat(){  console.log("吃吃吃");  }  }; |

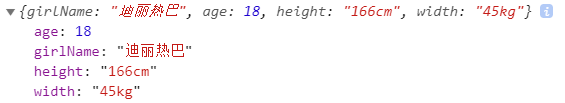
如果一个属性的值(ability)为函数，通常把这个属性称为“方法”。它可以像函数那样调用这个属性。frank.ability();

**1.4对象的创建**

直接使用大括号创建对象

|  |
| --- |
| var obj = {}; |

|  |
| --- |
| var girlFriend = {  girlName:"迪丽热巴",  age:18,  height:"166cm",  width:"45kg"  };  console.log(girlFriend); |



**1.5对象属性的读写**

1、读取属性

读取对象的属性,有两种方法:一种是使用点运算符,还有一种是使用方括号运算符。

语法：var obj = { key：value }；

console.log(obj.key);

console.log(obj['key']);

需要注意的是，使用方括号读取对象属性的时候需要加引号！

|  |
| --- |
| var girlFriend = {  girlName:"迪丽热巴",  age:18,  height:"166cm",  width:"45kg"  };  console.log(girlFriend.girlName);//迪丽热巴  console.log(girlFriend["age"]);//18 |

2、写入属性

点运算符和中括号运算符都能够给对象的属性赋值,无论属性是否存在(存在修改属性值，不存在增加属性)。

|  |
| --- |
| var girlFriend = {  girlName:"迪丽热巴",  age:18,  height:"166cm",  width:"45kg"  };  girlFriend.girlName = "古力娜扎";  girlFriend["where"] = "地球村";  console.log(girlFriend); |



中括号运算符总是能代替点运算符。但点运算符却不一定能全部代替中括号运算符。  
 （1）中括号运算符可以用字符串变量的内容作为属性名。点运算符不能。  
 （2）中括号运算符可以用纯数字为属性名。点运算符不能。

|  |
| --- |
| girlFriend["1height"] = "166cm";//正常  girlFriend.1height = "166cm";//报错 |

**1.6对象属性的操作**

1. 检查对象是否包含一个属性

|  |
| --- |
| 'key' in obj |

检查的是键名，不是键值。如果包含就返回true，否则返回false。

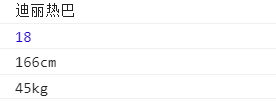
|  |
| --- |
| var girlFriend = {  girlName:"迪丽热巴",  age:18  };  console.log("age" in girlFriend);//true  console.log("height" in girlFriend);//false |

1. 遍历对象的所有属性

|  |
| --- |
| for (var variable in object)  {  执行的代码……  } |

循环变量 variable表示的是对象的属性名，并不表示属性的值！千万注意

|  |
| --- |
| var girlFriend = {  girlName:"迪丽热巴",  age:18,  height:"166cm",  width:"45kg"  };  for(var i in girlFriend){  console.log(girlFriend[i]);  } |

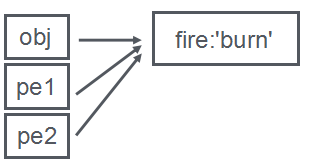


**1.7对象的引用**

如果不同的变量名指向同一个对象，那么它们都称为这个对象的引用。

也就是说这些变量共同指向同一个内存地址，修改其中一个变量所指向的值，都会影响到其他所有变量所指向的值。

|  |
| --- |
| var obj = { fire : 'burn' };  var people1\_fire = obj;  var people2\_fire = obj;  obj.fire = '灭了';  console.log(people1\_fire.fire);//灭了  console.log(people2\_fire.fire);//灭了 |



|  |
| --- |
| var a = 10;  var b = a;  console.log(a);//10  console.log(b);//10  a = 100;  console.log(a);//100  console.log(b);//10 |

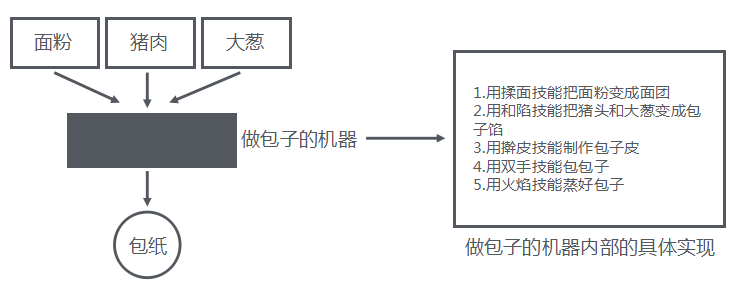
|  |
| --- |
| var obj1 = {  boyName:"李现",  age:18  };  var obj2 = obj1;  console.log(obj1);//{boyName: "李现", age: 18}  console.log(obj2);//{boyName: "李现", age: 18}  obj1.boyName = "张一山";  console.log(obj1);//{boyName: "张一山", age: 18}  console.log(obj2);//{boyName: "张一山", age: 18} |

1. **函数**

**2.1函数的概述**

官方：函数是由事件驱动的或者当它被调用时执行的可重复使用的代码块。

个人：函数是一组拥有某些特定功能的、能够被重复调用的、闭合代码块。(不权威)



上图中【做包子的机器】就可以被看做是一个函数。

有某些特定功能的：能包包子

能够被重复调用的：机器能使用多次

闭合：机器在外面看不到里面有什么

我们在函数外部并不关心函数里面有什么，只关心函数能做什么。(使用机器能够得到包子)

**函数的意义：在出现大量程序相同的时候，可以封装为一个function，这样只用调用一次，就能执行很多语句。**

**2.2函数的声明**

1. function命令法

|  |
| --- |
| function 函数名(参数1,参数2,…){  函数体  } |

**定义一个函数，用关键字function来定义**，function就是英语“功能”的意思。表示这里面定义的语句，完成了一些功能。function后面有一个空格，后面就是函数名字，函数的名字也是关键字，命名规范和变量命名是一样的。名字后面有一对儿圆括号，里面放置参数，我们一会儿介绍。然后就是大括号，大括号里面是函数的语句。

函数如果不调用，那么里面的语句就一辈子不会执行，不调用就等于白写。

|  |
| --- |
| function hello(){  console.log("hello");  } |

1. 函数表达式法

|  |
| --- |
| var 变量名 = function(参数1,参数2,…){  函数体  }; |

这种声明函数的方式没有函数名，而是用变量来指代函数。调用函数的时候通过访问变量名来调用函数。

|  |
| --- |
| var hello = function(){  console.log("hello");  }; |

**2.3函数的调用**

相较于函数声明的多种多样，函数的调用方式就简单很多。通常情况下只要函数已经被声明，直接写出函数名和函数参数即可调用函数。

|  |
| --- |
| 函数名(); |

一旦调用了函数，函数内部的语句就会执行。能够感觉到，函数是一些语句的集合，让语句成为一个军团，集体作战。要不出动都不出动，要出动就全出动。得到调用才出动。

|  |
| --- |
| function fun1(){  console.log(10);  }  function fun1(){  console.log(20);  }  fun1(); |

很显然通过调用能够看到，多次重复声明同名的函数会造成的结果就是：后声明的函数会覆盖先声明的函数。

对于javascript来说，把函数调用写在函数声明之前也是被允许的，因为javascript存在一个隐式的函数提升。

JS在执行前，会有一个预解析的过程，把所有的函数声明，都提升到了最最开头，然后再执行第一行语句。

所以，function定义在哪里，都不重要，程序总能找到这个函数。

|  |
| --- |
| fun1();  function fun1(){  console.log(10);  } |

**函数声明会被提升，但是函数表达式却不会被提升**

|  |
| --- |
| fun();  var fun = function(){ **//函数表达式，而不是function定义法**  alert("我是函数，我执行了！");  } |



**2.4函数的类型**

**2.4.1从定义角度分**

1. 内置函数：就是指JS自带的函数，如alert、prompt、parseInt等
2. 用户自定义函数：就是用户根据实际情况自己定义的函数

**2.4.2从参数角度分**

无参数、有参数

**2.4.3从返回值角度分**

无返回值函数、有返回值函数

**2.4.4其他**

回调函数、匿名函数、自执行函数

**2.5函数的参数**

定义在函数内部的语句，都是相同的，但是实际上我们可以通过“参数”这个东西，来让语句有差别。

定义函数的时候，内部语句可能有一些悬而未决的量，就是变量，这些变量，我们要求在定义的时候都罗列在小括号中：

|  |
| --- |
| function fun(a){  console.log("我第" + a + "次说爱你");  } |

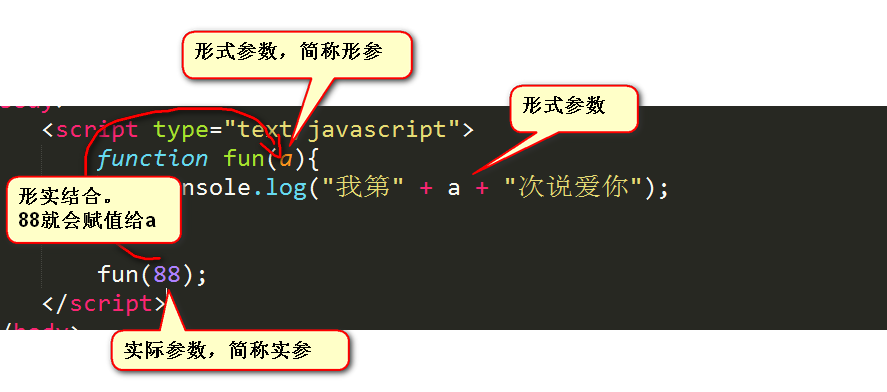
调用的时候，要把这个变量的真实的值，一起写在括号里，这样随着函数的调用，这个值也传给了a：

|  |
| --- |
| fun(88); |



形式参数：函数在定义的时候写在小括号中的参数被称为形式参数。形式参数只用来在函数内部使用，在函数外部形式参数失效。通常形式参数不用var声明，直接写变量名即可。

实际参数：函数在调用的时候写在小括号中的参数被称为实际参数。



参数可以有无数个，用逗号隔开。

|  |
| --- |
| //有多少形式参数都可以，都罗列出来  function fun(a**,**b){  console.log(a + b);  }  fun(3**,**5); //输出8  fun(8**,**11); //输出19 |

定义函数的时候，**参数是什么类型的没写，不需要指定类型**：

|  |
| --- |
| function sum(a,b){  console.log(a + b);  } |

也就是说调用的时候，**传进去什么什么类型，就是a、b什么类型**

|  |
| --- |
| sum("5",12); |

输出512，做的是连字符的运算。

我们还可以发现，**定义的时候和调用的时候参数个数可以不一样多，不报错**。

|  |
| --- |
| sum(10); |



因为我们只传了一个参数，b就没有传递，b被隐式的var了,所以值undefined。10+undefined就是NaN

|  |
| --- |
| sum(10,20,32,23,22,2,4); |



只有前两个参数被形参接收了，**后面的参数无视了**。

**2.6函数的返回值**

函数可以通过参数来接收东西，更可以通过return的语句来返回值，“吐出”东西。

|  |
| --- |
| function sum(a,b){  return a + b; //现在这个函数的返回值就是a+b的和  }    console.log(sum(3,8)); //sum没有输出功能，就要用console.log输出  //sum(3,8)实际上就成为了一个表达式，需要计算  //计算后就是11，console.log(11); |

程序遇见了return，将立即返回结果，返回调用它的地方，而不执行函数内的剩余的语句。

|  |
| --- |
| function fun(){  console.log(1);  console.log(2);  return; //返回一个空值  console.log(3); //这行语句不执行，因为函数已经return了  }  fun(); |



**2.7函数的作用域**

作用域（scope）指的是变量存在的范围。Javascript只有两种作用域：

全局作用域：变量在整个程序中一直存在，所有地方都可以读取。

函数作用域：变量只在函数内部存在。在函数外部变量失效。

和作用域与之对应的，javascript中有两种变量：

全局变量(global variable)：在函数外部声明的变量，它可以在函数内部读取。

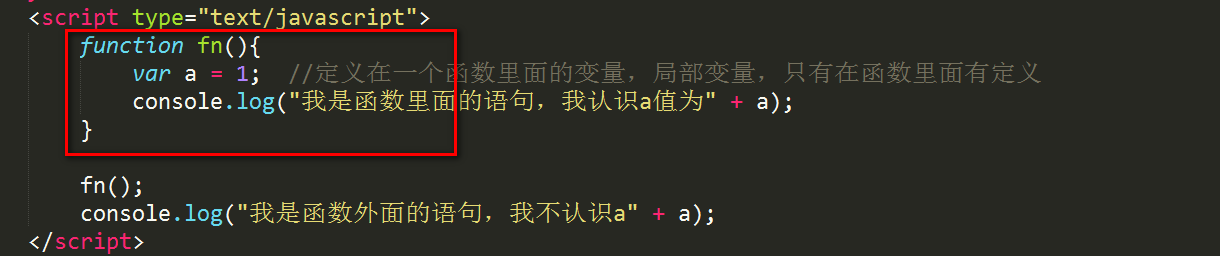
局部变量(local variable)：在函数内部定义的变量，外部无法读取。

**2.7.1函数能封闭住作用域**

一个变量如果定义在了一个function里面，那么这个变量就是一个局部变量，只在这个function里面有定义。出了这个function，就如同没有定义过一样。

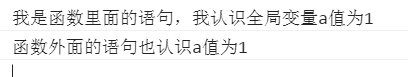
|  |
| --- |
| <script type="text/javascript">  function fn(){  var a = 1; //定义在一个函数里面的变量，局部变量，只有在函数里面有定义  console.log(**"我是函数里面的语句，我认识a值为" + a**);  }  fn();  console.log(**"我是函数外面的语句，我不认识a" + a**);  </script> |



a被var在了function里面，所以现在这个a变量只在红框范围内有定义：

如果一个变量，没有定义在任何的function中，那么它将在全部程序范围内都有定义：

|  |
| --- |
| var a = 1; //定义在全局范围内的一个变量，全局变量，在程序任何一个角落都有定义  function fn(){  console.log("我是函数里面的语句，我认识全局变量a值为" + a);  }  fn();  console.log("函数外面的语句也认识a值为" + a) |



总结一下：

● 在function里面直接用var声明的变量，叫做局部变量，只在function里面有定义，出了function没有定义的。

● 定义在全局范围内的，没写在任何function里面的，叫做全局变量，都认识。

**2.7.2不写var就自动成为全局变量**

|  |
| --- |
| function fn(){  a = 1; **//这个a第一次赋值的时候，并没有var过，**  **//所以就自动的在全局的范围帮你var了一次**  }  fn();  console.log(a); |

这是JS的一个机理，如果遇见了一个标识符，从来没有var过，并且还赋值了：

|  |
| --- |
| absdf = 123; |

那么就会自动帮你在全局范围内定义var absdf;

告诉我们一个道理，变量要老老实实写var。

**2.7.3函数的参数，会默认定义为这个函数的局部变量**

|  |
| --- |
| function fn(a,b,c,d){  } |

a,b,c,d就是一个fn内部的局部变量，出了fn就没有定义。

|  |
| --- |
| function fun(a){  console.log(a);  }  fun(10);  console.log(a); |



判断下列输出的结果：

|  |
| --- |
| function fun1 (a){//局部变量  console.log(a); //5  }  fun1(5);  console.log(a);//报错 |
| var a = 100;//全局  function fun1(){  console.log(a);//100  }  fun1();  console.log(a);//100 |
| var a = 100;//全局  function fun1(a){//局部  console.log(a);//50  }  fun1(50);  console.log(a);//100 |
| var a = 100;//全局  function fun1(){  a = 50;//全局  console.log(a);//50  }  fun1();  console.log(a);//50 |
| var a = 100;//全局  function fun1(){  var a = 50;//局部  console.log(a);//50  }  fun1();  console.log(a);//100 |
| var a = 100;//全局变量  function fun1(){  a++;//全局变量 a = a+1  console.log(a);//101  }  fun1();  console.log(a);//101 |
| var a = 100;//全局变量  function fun1(a){//局部变量  a++;//局部变量  console.log(a);//101  }  fun1(100);  console.log(a);//100 |
| var a;//全局  var b;//全局  var c;//全局  function fun1(){  var a = 100;//局部  var b = 100;//局部  var c = 100;//局部  console.log(a);//100  console.log(b);//100  console.log(c);//100  }  fun1();  console.log(a);//undefined  console.log(b);//undefined  console.log(c);//undefined |
| var a;//全局  var b;//全局  var c;//全局  function fun1(){  a = 100;//全局  b = 100;//全局  c = 100;//全局  console.log(a);//100  console.log(b);//100  console.log(c);//100  }  fun1();  console.log(a);//100  console.log(b);//100  console.log(c);//100 |
| var a;//全局  var b;//全局  var c;//全局  function fun1(){  var a = b = c = 100;//a 局部 b c 全局  console.log(a);//100  console.log(b);//100  console.log(c);//100  }  fun1();  console.log(a);//undefined  console.log(b);//100  console.log(c);//100 |