# DAY05 对象和数组

2023年4月24日 18:04

## 作用域和作用域链

### 1.作用域:

就是一个变量可以生效的范围

变量不是在所有地方都可以使用的，而这个变量的使用范围就是作用域

### 2.全局作用域

全局作用域是最大的作用域.

在全局作用域中定义的变量可以在任何地方使用.

这个作用域会一直存在(弊端)，直到页面关闭就销毁了

### 3.局部作用域 - 函数内部的作用域

局部作用域就是在全局作用域下面又开辟出来的一个相对小一些的作用域

在JS中只有函数能生成一个局部作用域，别的都不行

每一个函数，都是一个局部作用域

重点：在局部作用域中定义的变量只能在这个局部作用域内部使用

```
// var a = 1;//全局变量
// var b = 2;
// function fn() {
//    var c = 3;//局部变量
//    var d = 4;
//    c++;
//    d++;
//    console.log(a);//1
//    console.log(b);//2
//    console.log(c);//4
//    console.log(d);//5
// }
// fn();
// console.log(a);//1
// console.log(c);//报错
```

### 4.特殊的作用域问题(如果函数内部声明变量没有添加var，变成全局作用域)

声明变量可以不加var,虽然不推荐这样做，但是确实可以省略var

```
// a = 10;
// console.log(a);//10
// function fn() {
//    a = 1;//声明变量省略了var，变成全局的变量
//    b = 10;//声明变量省略了var，变成全局的变量
// }
```

```
// fn();
// console.log(b);//10
// console.log(a);//1
```

**5.作用域链**

通俗地讲，当执行函数时，如何获取变量的值

首先，在自己的作用域内部查找，如果有，就直接拿来使用

如果没有，就去上一级作用域查找，如果有，就拿来使用

如果没有，就继续去上一级作用域查找，依次类推

如果一直到全局作用域都没有这个变量，那么就会直接报错（该变量 is not defined）

```
var num = 1;
function fn1() {
  var num = 2;
  function fn2() {
    var num = 3;
    function fn3() {
      var num = 4;
      console.log(num);
    }
    fn3()
  }
  fn2();
}
fn1();
```

递归函数

函数调用函数自身

1.在编程世界里面，递归函数是在一个函数通过名字调用自身情况下构成的，即**函数内部调用函数本身**

**2.递归需要有边界条件、递归前进段和递归返回段**

当边界条件不满足时，递归前进；当边界条件满足时，递归返回

在使用递归策略时，必须有一个明确的递归结束条件，称为递归出口

```
// 递：传递
// 归：返回，回来
// 案例:求阶乘
// 5! = 5 * 4 * 3 * 2 * 1 ==> 5 * 4!
// 4! = 4 * 3!
// 3! = 3 * 2!
// 2! = 2 * 1!
// 1! = 1     // 递归结束条件，递归出口
function jc(n) {// n = 5
  if (n === 1) {// 递归结束条件，递归出口
    return 1;
```

```
    }
    return n * jc(n - 1);
  //第一次：5 * jc(4)  5*4*3*2*1
  //第二次：4 * jc(3)  4*3*2*1
  //第三次：3 * jc(2)  3*2*1
  //第四次：2 * jc(1)  2*1
  //jc(1) 满足递归出口  返回1
}
console.log(jc(5));//120
// Maximum call stack size exceeded : 栈溢出错误，内存溢出错误，
```

**递归的缺点**

递归与循环有着类似的思想，但我们很少用，因为其运行效率较低 、可能出现死循环(栈溢出)。

因此，应该尽量避免使用递归，除非没有更好的算法或者某种特定情况，递归更为适合的时候。

**构造函数及函数的细分**

**一.函数的声明方式有两种**

1.声明式函数：普通函数，有名称的函数

2.赋值式函数：函数表达式，没有名称的函数

```
var fn = function(){}
box1.onclick = function () {console.log('函数触发了11111');}
!function(){}()
```

**二.赋值式函数的扩展(函数表达式的扩展)**

1.事件处理函数：将函数通过事件去使用(可以使用普通函数，也可以使用函数表达式)。

```
// function fn() {
//   console.log('函数触发了');
// }
// box.onclick = fn();//这样写是不合理的，这样其实是将函数的调用后的返回值给box.onclick
// box.onclick = fn;//这样写将函数交给事件(被动的方法)来进行调用
// 函数名等同于整个的函数结构体
```

2.自执行函数：将函数的声明和调用写到一起，形成了函数表达式，然后具有一次性。

函数名等同于整个的函数结构体

2.1.下面的写法，语法通不过，原因是没有将整个函数当作整体

```
// function fn() {
```

```
//   console.log('我是函数111111111111');
// } ();
// (function () {
//   console.log('我是函数111111111111');
// })();
```

2.2.自执行函数的其他写法

!可以将函数变成函数表达式，还有一些其他符号也可以实现，比如：+ - ~ ！ ()

使用最多的是!和()

```
// !function () {
//   console.log('我是函数222222222');
// }();
```

2.3.自执行函数的特点

具有一次性,无法重用

对作用域有好处的，利用自执行函数来形成局部的作用域，不会造成冲突。

**三.提前了解构造函数**

构造函数(类)：一类事物，抽象的，不占据内存空间，用来生成对象(实例对象)。

构造函数也是函数，ES里面的构造函数和类是一种意义。

1.构造函数的特点

1.1.首字母大写

1.2.不是直接调用，必须通过new关键字去调用。

注意：当前使用的所有构造函数都是由系统提供，直接使用生成实例对象即可。

```
// function Fn() {
//   console.log('11111111');
// }
// var f1 = Fn();  // Fn是普通函数  f1是函数的返回值
// console.log(f1); //undefined
// var f2 = new Fn();// Fn是构造函数   f2是对象(实例对象)
// console.log(f2); //f2是一个对象
```

**四.页面打开的时候，浏览器会自动给我们生成一个全局作用域window**

window叫做全局对象，自动生成的

1.var声明的变量属于全局对象window下面的属性

```
// var num = 100;
// console.log(num);//100
// console.log(window.num);//100
```

2.函数属于全局对象window下面的方法

```
// function fn() {
//   console.log('我是函数');
// }
// fn();
// window.fn();
```

**数组对象的概述以及创建数组对象**

**一.数组对象的概述**

数组是一组相同类型或者不同类型的数据。

除了Object类型之外，Array类型是ES最常用的类型。

ES中的数组每个数组项可以保存任何类型。ES中数组的大小也是可以调整的。

**二.数组对象的创建**

1.利用系统提供的Array构造函数创建

```
// var arr = new Array(1, 2, 3, 'zhangsan', 'lisi', true);//arr:数组对象,实例对象
// console.log(typeof arr);//object
// console.log(arr);// [1, 2, 3, 'zhangsan', 'lisi', true]
```

2.字面量创建数组对象 - 推荐的方式

在计算机科学中，字面量是用于表达源代码中一个固定值的表示法，创建数组的简洁方式

```
// var arr = [1, 2, 3, 'zhangsan', 'lisi', true];
// console.log(typeof arr);//object
// console.log(arr);// [1, 2, 3, 'zhangsan', 'lisi', true]
```

小小的区别

如果数组里面只有一项值，而且这一项值是数字，表示的是长度。

```
// var a1 = new Array(20);  //20是数组的长度，不是数组项的值
// var a2 = [20];//20是数组项的值
// console.log(a1.length);//20
// console.log(a2.length);//1
```

**数组的相关操作**

**一.数组的操作**

1.索引下标：数组使用索引(index)下标来读取数组项的值，索引下标是固定的，从0开始的整数。

```
// var arr = ['apple', 'banana', 'pear', 'orange'];
// console.log(arr[0]);//apple
// console.log(arr[1]);//banana
```

2.数组length属性：代表数组的长度，通过设置这个属性，可以从数组的末尾移除项或向数组

中添加新项。

```
// var arr = ['apple', 'banana', 'pear', 'orange'];
// console.log(arr.length);//4
```

注意：清空数组直接将length属性设为0

```
// arr.length = 0;
// console.log(arr);//[]
```

3.数组的遍历：获取所有数组项的值，数组的索引是有序的整数，可以采用for循环进行遍历。

遍历：获取数组的每一项值

```
// var arr = ['apple', 'banana', 'pear', 'orange', 'apple', 'banana',
'pear', 'orange'];
// for (var i = 0; i < arr.length; i++) {
//   console.log(arr[i]);
// }
```

4.检测数组的方法

Array.isArray()：确定括号中的某个值到底是不是数组,返回布尔值

方法以构造函数名开头的，叫做静态方法

```
// var a = null;
// var b = [1, 2, 3, 4];
// console.log(typeof a);//object
// console.log(typeof b);//object
// console.log(Array.isArray(a));//false 不是数组
// console.log(Array.isArray(b));//true 是数组
```

**自定义对象的概念以及创建**

**一.自定义对象的概念**

概念：

对象是ES的数据类型,对象是一种复合值,它将很多值聚合在一起,可通过名字访问这些值

对象也看做是属性的无序集合,每个属性都是一个键/值对。

问题1：为什么对象是属性的无序集合，因为当属性值是函数的时候，可以叫方法。

问题2：键值对，css属性词的格式就是键值对，key:value

总结：利用Object构造函数给生成的对象添加自定义的属性或者方法。

**二.对象的二种创建方式**

1.利用Object构造函数

```
var obj = new Object();//obj:自定义对象，实例对象     Object：构造函数(类)
```

```
// 给自定义对象添加属性和方法
obj.name = 'zhangsan';
obj.age = 18;
obj.sex = '男';
obj.show = function () {//属性值是函数，叫方法
  return `我的名字叫${obj.name},我今年${obj.age}岁,我是${obj.sex}的`;
};
console.log(obj);
// 读属性的值
console.log(obj.name);//zhangsan
console.log(obj.age);//18
console.log(obj.sex);//男
// 读方法
console.log(obj.show());
```

2.利用字面量创建

```
var obj1 = {
  name: 'zhangsan',
  age: 18,
  sex: '男',
  show: function () {
    return `我的名字叫${obj1.name},我今年${obj1.age}岁,我是${obj1.sex}的`;
  }
};
console.log(obj1);
// 读属性的值
console.log(obj1.name);//zhangsan
console.log(obj1.age);//18
console.log(obj1.sex);//男
// 读方法
console.log(obj1.show());
```

**自定义对象的操作以及特点**

**一.自定义对象的操作**

创建对象

```
var a = 'address';
var obj = {
  sex: '男'
};
```

1.增

注意：点符号和中括号都可以添加属性或者读取属性值，区别是中括号里面可以放变量或者字

符串，点后面只能跟字符串(无需添加引号)。

```
obj.name = 'zhangsan';
obj.age = 20;
obj['grade'] = '2303班';
obj[a] = '浙江杭州九堡';// a是上面的变量名
```

2.删

```
// delete obj.name;
// delete obj.age;
```

3.改

属性名相同，下面的覆盖上面，

```
obj.name = 'wangwu';//覆盖上面
obj['name'] = '尼古拉斯赵四';//覆盖上面
```

4.查

属性名不存在，不会报错，输出undefined

```
// console.log(obj.address);//浙江杭州九堡
// console.log(obj['name']);//尼古拉斯赵四
// console.log(obj.hehehehehehehehe);//undefined,属性名不存在，不会报错
// console.log(obj);
```

**二.自定义对象的特点 - 非常重要**

1.对象的属性名一定是字符串格式。

2.可以通过点符号和中括号来操作对象的属性，但是点后面只能跟字符串(无需添加引号),中括号里面可以是变量字符串数字等。

3.属性名相同，下面的覆盖上面，属性名不存在，不会报错，输出undefined

4.整个对象的值都是一样的[object Object]

```
var obj1 = {
  1: 'zhangsan',
  2: 'lisi'
}
var obj2 = {
  3: 'zhangsan',
  4: 'lisi',
  5: 'wangwu'
}
```

```
console.log(obj1);//输出对象的结构
console.log(obj2);//输出对象的结构
// 通过alert方法输出对象的值
alert(obj1)//[object Object]
alert(obj2)//[object Object]
```

**对象的遍历以及静态方法**

**一.对象的遍历(获取对象里面每一个属性值)**

对象是属性的无序集合，不能使用for循环。

1.使用 for in循环遍历对象

for (var 变量 in 对象名) {

会根据对象内有多少个成员(属性方法)执行多少回

随着循环, 变量 分别表示对象内的每一个 key(属性名)

随着循环, obj[变量] 分别表示对象内的每一个 值

}

```
// var obj = {
//   name: 'zhangsan',
//   age: 18,
//   sex: '女'
// };
// for (var key in obj) {//key:属性名   obj:遍历的对象
//   // console.log(key);//每一个属性名
//   console.log(obj[key]);//获取属性名对应的值，必须采用中括号，中括号里面可以放变量，字符串，数字
// }
```

2.for in也可以遍历数组，不推荐。

```
// var arr = ['zhangsan', 'lisi', 'wangwu'];
// for (var index in arr) {//index:数组项的索引   arr: 遍历的数组
//   // console.log(index);//获取索引
//   console.log(arr[index]);//获取索引对应的值
// }
```

**二.对象下面的静态方法(以构造函数名开头的，绑定在构造函数下面的方法)**

```
// var obj = {
//   name: 'zhangsan',
//   age: 18,
//   sex: '女'
// };
```

1.Object.keys(): 将对象的key值放到一个数组中返回。

```
// console.log(Object.keys(obj));//['name','age','sex']
```

2.Object.values(): 将对象的value值放到一个数组中返回。

```
// console.log(Object.values(obj));//['zhangsan',18,'女']
```

**数据类型之间存储的区别**

## 一.数据类型之间存储的区别

数据类型分为两大类

1.基本类型：number/string/boolean/null/undefined/Symbol/BigInt

2.引用类型：object

## 二.基本类型在内存中的存储情况

### 1.内存的划分

1.1.栈区：主要存储基本数据类型的内容和引用类型的地址

1.2.堆区：主要存储引用类型的内容

### 2.基本类型和引用类型存储方式

2.1.基本类型内容和地址都在栈区，直接在栈空间内有存储一个数据

2.2.引用类型地址在栈区，内容在堆区

在堆里面开辟一个存储空间

把数据存储到存储空间内

把存储空间的地址赋值给栈里面的变量

## 三.基本类型和引用类型的赋值

### 1.基本类型的赋值(值传递)

```
// var a = 10;
// var b = a;//将a的值复制一份，传递给b
// b++;
// console.log(a);//10
// console.log(b);//11
```

### 2.引用类型的赋值(引用传递，地址传递)

```
 // var arr1 = [1, 2, 3];
 // var arr2 = arr1;//将arr1的地址传递给arr2,arr1和arr2指向同一个地址，arr2改变地址里面的内容，arr1肯定要改变。
 // arr2.length = 0;
 // console.log(arr1);//[]
```

```
// console.log(arr2);//[]
```

注意：引用类型传递的是地址，涉及到引用类型的赋值都是在传递地址，最终都会指向同一个内容区。

## 四.数据类型之间的比较

1.基本类型的比较,比较的就是值

```
// var n1 = 10;
// var n2 = 10;
// console.log(n1 === n2);//true
```

2.引用类型的比较，比较的是地址

```
// var arr1 = [];//创建对象，产生地址以及地址对应的内容
// var arr2 = [];//创建对象，产生地址以及地址对应的内容
// console.log(arr1 === arr2);//false  比较的是地址，两个对象对应两个地址
// console.log({} === {});//false 比较的是地址，两个对象对应两个地址
```