对象

讲师：李立超

Object对象

- Object类型，我们也称为一个对象。是JavaScript中的引用数据类型。
- 它是一种复合值，它将很多值聚合到一起，可以通过名字访问这些值。
- 对象也可以看做是属性的无序集合，每个属性都是一个名/值对。
- 对象除了可以创建自有属性，还可以通过从一个名为原型的对象那里继承属性。
- 除了字符串、数字、true、false、null和undefined之外，JS中的值都是对象。

创建对象

- 创建对象有两种方式：
  - 第一种

对象构造函数

```
var person = new Object();
person.name = "孙悟空";
person.age = 18;
```

  - 第二种

对象字面量

```
var person = {
        name:"孙悟空",
        age:18
};
```

对象属性的访问

- 访问属性的两种方式：
  - .访问
    - 对象.属性名
  - []访问
    - 对象[‘属性名’]

基本数据类型

- JS中的变量可能包含两种不同数据类型的值：基本数据类型和引用数据类型。
- JS中一共有5种基本数据类型：String、Number、Boolean、Undefined、Null。
- 基本数据类型的值是无法修改的，是不可变的。
- 基本数据类型的比较是值的比较，也就是只要两个变量的值相等，我们就认为这两个变量相等。

引用数据类型

- 引用类型的值是保存在内存中的对象。
- 当一个变量是一个对象时，实际上变量中保存的并不是对象本身，而是对象的引用。
- 当从一个变量向另一个变量复制引用类型的值时，会将对象的引用复制到变量中，并不是创建一个新的对象。
- 这时，两个变量指向的是同一个对象。因此，改变其中一个变量会影响另一个。

栈和堆

- JavaScript在运行时数据是保存到栈内存和堆内存当中的。
- 简单来说栈内存用来保存变量和基本类型。堆内存用来保存对象。
- 我们在声明一个变量时实际上就是在栈内存中创建了一个空间用来保存变量。
- 如果是基本类型则在栈内存中直接保存，
- 如果是引用类型则会在堆内存中保存，变量中保存的实际上对象在堆内存中的地址。

栈和堆

```
var a = 123;
var b = true;
var c = "hello";
var d = {name:'sunwukong',age:18};
```

栈内存	
d	0x000
c	hello
b	true
a	123

堆内存	
0x000	name = ‘sunwukong’ age = 18

数组

- 数组也是对象的一种。
- 数组是一种用于表达有顺序关系的值的集合的语言结构。
- 创建数组：
  - var array = [1,44,33];
- 数组内的各个值被称作元素。每一个元素都可以通过索引（下标）来快速读取。索引是从零开始的整数。

函数

- 函数是由一连串的子程序（语句的集合）所组成的，可以被外部程序调用。向函数传递参数之后，函数可以返回一定的值。
- 通常情况下，JavaScript 代码是自上而下执行的，不过函数体内部的代码则不是这样。如果只是对函数进行了声明，其中的代码并不会执行。只有在调用函数时才会执行函数体内部的代码。
- 这里要注意的是JavaScript中的函数也是一个对象。

函数的声明（一）

- 首先明确一点函数也是一个对象，所以函数也是在堆内存中保存的。
- 函数声明比较特殊，需要使用function关键字声明。

```
var sum = function(a,b){return a+b};
```

- 上边的例子就是创建了一个函数对象，并将函数对象赋值给了sum这个变量。其中()中的内容表示执行函数时需要的参数，{}中的内容表示函数的主体。

函数的调用

- 调用函数时，传递给函数的参数称为实参（实际参数）。
- 如果想调用我们上边定义的sum函数，可以这样写：

```
var result = sum(123,456);
```

  - 这样表示调用sum这个函数，并将123和456作为实参传递给函数，函数中会将两个参数求和并赋值给result。

函数的声明（二）

- 可以通过函数声明语句来定义一个函数。函数声明语句以关键字 function 开始，其后跟有函数名、参数列表和函数体。其语法如下所示：

```
function 函数名(参数,参数,参数...){
        函数体
}
```

- 例如:

```
function sum(a,b){
        return a+b;
}
```

  - 上边我们定义了一个函数名为sum，两个参数a和b。函数声明时设置的参数称为形参（形式参数），这个函数对两个参数做了加法运算并将结果返回。

传递参数

- JS中的所有的参数传递都是按值传递的。也就是说把函数外部的值赋值给函数内部的参数，就和把值从一个变量赋值给另一个变量是一样的。

执行环境

- 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，环境中定义的所有变量和函数都保存在这个对象中。
- 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。
- 某个执行环境中的所有代码执行完毕后，该环境被销毁，保存在其中的所有变量和函数定义也随之销毁。
- 在内部环境可以读取外部环境的变量，反之则不行。

函数内部属性

- 在函数内部，有两个特殊的对象：
  - arguments
    - 该对象实际上是一个数组，用于保存函数的参数。
    - 同时该对象还有一个属性callee来表示当前函数。
  - this
    - this 引用的是一个对象。对于最外层代码与函数内部的情况，其引用目标是不同的。
    - 此外，即使在函数内部，根据函数调用方式的不同，引用对象也会有所不同。需要注意的是，this 引用会根据代码的上下文语境自动改变其引用对象。

this 引用的规则

- 在最外层代码中，this 引用的是全局对象。
- 在函数内，this 根据函数调用方式的不同而有所不同：

函数的调用方式	this引用的对象
构造函数	所生成的对象
调用对象的方法	当前对象
apply或call调用	参数指定的对象
其他方式	全局对象（window）

构造函数

- 构造函数是用于生成对象的函数，像之前调用的Object()就是一个构造函数。
- 创建一个构造函数：

```
function MyClass(x,y) {
    this.x = x;
    this.y = y;
}
```

- 调用构造函数：
  - 构造函数本身和普通的函数声明形式相同。
  - 构造函数通过 new 关键字来调用，new 关键字会新创建一个对象并返回。
  - 通过 new关键字调用的构造函数内的 this 引用引用了（被新生成的）对象。

new关键字

- 使用new关键字执行一个构造函数时：
  - 首先，会先创建一个空的对象。
  - 然后，会执行相应的构造函数。构造函数中的this将会引用这个新对象。
  - 最后，将对象作为执行结果返回。
- 构造函数总是由new关键字调用。
- 构造函数和普通函数的区别就在于调用方式的不同。
- 任何函数都可以通过new来调用，所以函数都可以是构造函数。
- 在开发中，通常会区分用于执行的函数和构造函数。
- 构造函数的首字母要大写。

属性的访问

- 在对象中保存的数据或者说是变量，我们称为是一个对象的属性。
- 读取对象的属性有两种方式：
  - 对象.属性名
  - 对象['属性名']
- 修改属性值也很简单：
  - 对象.属性名 = 属性值
- 删除属性
  - delete 对象.属性名
- constructor
  - 每个对象中都有一个constructor属性，它引用了当前对象的构造函数。

垃圾回收

- 不再使用的对象的内存将会自动回收，这种功能称作垃圾回收。
- 所谓不再使用的对象，指的是没有被任何一个属性（变量）引用的对象。
- 垃圾回收的目的是，使开发者不必为对象的生命周期管理花费太多精力。

原型继承

- JS是一门面向对象的语言，而且它还是一个基于原型的面向对象的语言。
- 所谓的原型实际上指的是，在构造函数中存在着一个名为原型的(prototype)对象，这个对象中保存着一些属性，凡是通过该构造函数创建的对象都可以访问存在于原型中的属性。
- 最典型的原型中的属性就是toString()函数，实际上我们的对象中并没有定义这个函数，但是却可以调用，那是因为这个函数存在于Object对应的原型中。

设置原型

- 原型就是一个对象，和其他对象没有任何区别，可以通过构造函数来获取原型对象。
  - 构造函数. prototype
- 和其他对象一样我们可以添加修改删除原型中的属性，也可以修改原型对象的引用。
- 需要注意的是prototype属性只存在于函数对象中，其他对象是没有prototype属性的。
- 每一个对象都有原型，包括原型对象也有原型。特殊的是Object的原型对象没有原型。

获取原型对象的方法

- 除了可以通过构造函数获取原型对象以外，还可以通过具体的对象来获取原型对象。
  - Object.getPrototypeOf(对象)
  - 对象.__proto__
  - 对象. constructor.prototype
- 需要注意的是，我们可以获取到Object的原型对象，也可以对它的属性进行操作，但是我们不能修改Object原型对象的引用。

原型链

- 基于我们上边所说的，每个对象都有原型对象，原型对象也有原型对象。
- 由此，我们的对象，和对象的原型，以及原型的原型，就构成了一个原型链。
- 比如这么一个对象：
  - var mc = new MyClass(123,456);
  - 这个对象本身，原型MyClass.proprototype原型对象的原型对象是Object，Object对象还有其原型。这组对象就构成了一个原型链。
  - 这个链的次序是：mc对象、mc对象原型、原型的原型（Object）、Object的原型
- 当从一个对象中获取属性时，会首先从当前对象中查找，如果没有则顺着向上查找原型对象，直到找到Object对象的原型位置，找到则返回，找不到则返回undefined。

instanceof

- 之前学习基本数据类型时我们学习了typeof用来检查一个变量的类型。
- 但是typeof对于对象来说却不是那么好用，因为任何对象使用typeof都会返回Object。而我们想要获取的是对象的具体类型。
- 这时就需要使用instanceof运算符了，它主要用来检查一个对象的具体类型。
- 语法：
  - var result = 变量 instanceof 类型

引用类型

- 上边我们说到JS中除了5种基本数据类型以外其余的全都是对象，也就是引用数据类型。
- 但是虽然全都是对象，但是对象的种类却是非常繁多的。比如我们说过的Array（数组），Function（函数）这些都是不同的类型对象。
- 实际上在JavaScript中还提供了多种不同类型的对象。

Object

- 目前为止，我们看到的最多的类型就是Object，它也是我们在JS中使用的最多的对象。
- 虽然Object对象中并没有为我们提供太多的功能，但是我们会经常会用途来存储和传输数据。
- 创建Object对象有两种方式：
  - var obj = new Object();
  - var obj = {}
- 上边的两种方式都可以返回一个Object对象。
- 但是第一种我们使用了一个new关键字和一个Object()函数。
- 这个函数就是专门用来创建一个Object对象并返回的，像这种函数我们称为构造函数。

Array

- Array用于表示一个有序的数组。
- JS的数组中可以保存任意类型的数据。
- 创建一个数组的方式有两种：
  - 使用构造器：
    - var arr = new Array(数组的长度);
    - var arr = new Array(123,’hello’,true);
  - 使用[]
    - var arr = [];
    - var arr = [123,’hello’,false];
- 读取数组中的值使用数组[索引]的方式，注意索引是从0开始的。

Date

- Date类型用来表示一个时间。
- Date采取的是时间戳的形式表示时间，所谓的时间戳指的是从1970年1月1日0时0秒0分开始经过的毫秒数来计算时间。
- 直接使用new Date()就可以创建一个Date对象。
- 创造对象时不传参数默认创建当前时间。可以传递一个毫秒数用来创建具体的时间。
- 也可以传递一个日期的字符串，来创建一个时间。
  - 格式为：月份/日/年 时:分:秒
  - 例如：06/13/2004 12:12:12

Function

- Function类型代表一个函数，每一个函数都是一个Function类型的对象。而且都与其他引用类型一样具有属性和方法。
- 由于函数是对象，因此函数名实际上也是一个指向函数对象的指针，不会与某个函数绑定。
- 函数的声明有两种方式：
  - function sum(){}
  - var sum = function(){};
- 由于存在函数声明提升的过程，第一种方式在函数声明之前就可以调用函数，而第二种不行。

函数也可以作为参数

- 函数也是一个对象，所以函数和其他对象一样也可以作为一个参数传递给另外一个函数。
- 但是要注意的是使用函数作为参数时，变量后边千万不要加()，不加()表示将函数本身作为参数，加上以后表示将函数执行的结果作为参数。

函数对象的方法

- 每个函数都有两个方法call()和apply()。
- call()和apply()都可以指定一个函数的运行环境对象，换句话说就是设置函数执行时的this值。
- 使用方式：
  - 函数对象.call(this对象,参数数组)
  - 函数对象.apply(this对象,参数1,参数2,参数N)

闭包（closure）

- 闭包是JS一个非常重要的特性，这意味着当前作用域总是能够访问外部作用域中的变量。因为函数是JS中唯一拥有自身作用域的结构，因此闭包的创建依赖于函数。
- 也可以将闭包的特征理解为，其相关的局部变量在函数调用结束之后将会继续存在。

基本包装类型

- 基本数据类型是不能去调用方法的，所以JS中还提供了3个特殊的引用类型：
  - Boolean
  - Number
  - String
- 这三个类型分别包装了Boolean、Number、String并扩展了许多实用的方法。
- 他们的使用方式和普通的对象一样。
- 要注意的是使用typeof检查这些包装类型时返回的都是object。

Boolean

- Boolean 类型是与布尔值对应的引用类型。
- 可以采用这种方式创建：
  - var booleanObject = new Boolean(true);
- 我们最好永远不要使用Boolean包装类。

Number

- Number是数值对应的引用数据类型。创建Number对象只需要在调用构造函数时传递一个数值：
  - var num = new Numbaer(20);
- 使用数值时我们建议使用基本数值，而不建议使用包装类。

String

- String 类型是字符串的对象包装类型，可以像下面这样使用 String 构造函数来创建。
  - var str = new String("hello world");
- 可以使用length属性来获取字符串的长度。

Math

- JS 还为保存数学公式和信息提供了一个公共位置，即 Math 对象。
- 与我们在 JavaScript 直接编写的计算功能相比， Math 对象提供的计算功能执行起来要快得多。 Math 对象中还提供了辅助完成这些计算的属性和方法。

Math对象的属性

属性	说明
Math.E	自然对数的底数，即常量 e 的值
Math.LN10	10的自然对数
Math.LN2	2的自然对数
Math.LOG2E	以2为底 e 的对数
Math.LOG10E	以10为底 e 的对数
Math.PI	π的值
Math.SQRT1_2	1/2的平方根（即2的平方根的倒数）
Math.SQRT2	2的平方根

Math的方法

- 最大最小值
  - Math.max()获取最大值
  - Math.min()获取最小值
- 舍入：
  - 向上舍 Math.ceil()
  - 向下舍 Math.floor()
  - 四舍五入 Math.round()
- 随机数： Math.random()
  - 选取某个范围内的随机值：
    - 值 = Math.floor(Math.random() * 可能值的总数 + 第一个可能的值)


尚硅谷
www.atguigu.com