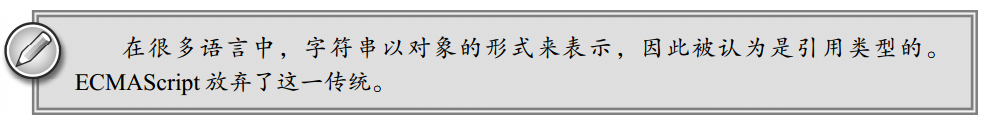
# 基本类型和引用类型

ECMAScript 变量可能包含两种不同数据类型的值：基本类型值和引用类型值。 基本类型值指的是简单的数据段，而引用类型值指那些可能由多个值构成的对象。

在将一个值赋给变量时，解析器必须确定这个值是基本类型值还是引用类型值。 5 种基本数据类型： Undefined、 Null、 Boolean、 Number 和 String。这 5 种基本数据类型是按值访问的，因为可以操作保存在变量中的实际的值。

引用类型的值是保存在内存中的对象。与其他语言不同， JavaScript 不允许直接访问内存中的位置，也就是说不能直接操作对象的内存空间。在操作对象时，实际上是在操作对象的引用而不是实际的对象。

为此，引用类型的值是按引用访问的。



# **数组入门**

什么是数组呢？简而言之，它就是一个用于存储数据的列表。与一次只能存储一个数据的变量不同，我们可以用数组来存储任意数量的元素值。

## **创建数组**

我们可以用一对不带任何内容的方括号来声明一个空数组变量，例如：

|  |
| --- |
| > var a = []; |

如果我们想要定义一个带三个元素的数组，则可以这样做：

|  |
| --- |
| > var a = [1,2,3]; |

只要在控制台中输入相应的数组名，就能打印出该数组中的所有内容：

|  |
| --- |
| > a;  [1, 2, 3] |

## **数组元素访问**

现在的问题是，我们应该如何访问数组中的各个数据元素呢？ 通常，元素在数组中的索引位置（下标）是从 0 开始编号的。也就是说，数组首元素的索引值（或者说位置值）应该是 0，第二个元素的索引值则是 1，以此类推。

为了访问特定的数组元素，我们需要用一对方括号来指定元素的索引值。因此 a[0]所

访问的就是数组 a 的首元素，而 a[1]则代表第二个元素，以此类推。

|  |
| --- |
| > a[0];  1  > a[1];  2 |

## **增加、更新数组元素**

我们可以通过索引来更新数组中的元素。例如在下面的代码中，我们更新了第三个元

素（索引值为 2）的值，并将更新后的数组打印出来：

|  |
| --- |
| > a[2] = 'three';  "three"  > a;  [1, 2, "three"] |

另外，我们也可以通过索引一个之前不存在的位置，来为其添加更多的数组元素。

|  |
| --- |
| > a[3] = 'four';  "four"  > a;  [1, 2, "three", "four"] |

如果新元素被添加的位置与原数组末端之间存在一定的间隔，那么这之间的元素将会

被自动设定为 undefined 值。例如：

|  |
| --- |
| > var a = [1,2,3];  > a[6] = 'new';  "new"  > a;  [1, 2, 3, undefined x 3, "new"] |

## **删除元素**

为了删除特定的元素，我们需要用到 delete 操作符。然而，相关元素被删除后，原

数组的长度并不会受到影响。从某种意义上来说，该元素被删除的位置只是被留空了而已。

|  |
| --- |
| > var a = [1, 2, 3];  > delete a[1];  true  > a;  [1, undefined, 3]  > typeof a[1];  "undefined" |

## **数组的数组**

我们可以在数组中存放任何类型的值，当然也包括另一个数组。

|  |
| --- |
| > var a = [1, "two", false, null, undefined];  > a;  [1, "two", false, null, undefined]  > a[5] = [1,2,3];  [1, 2, 3]  > a;  [1, "two", false, null, undefined, Array[3]] |

如果我们用鼠标单击控制台内结果里的 Array[3]，这个数组的值就会被展开。下面

我们再来看另一个例子，这里定义了一个含有两个数组的数组：

|  |
| --- |
| > var a = [[1,2,3],[4,5,6]];  > a;  ［ Array[3],Array[3]］ |

在该数组中，首元素 a[0]本身也是一个数组。

|  |
| --- |
| > a[0];  [1, 2, 3] |

所以如果想要访问内层数组中的特定元素，我们就得要再加一组方括号。例如：

|  |
| --- |
| > a[0][0];  1  > a[1][2];  6 |

值得注意的是，我们也可以通过这种数组访问方式来获取字符串中特定位置上的字符。

例如：

|  |
| --- |
| > var s = 'one';  > s[0];  "o"  > s[1];  "n"  > s[2];  "e" |

## **数组遍历**

### **普通for循环**

经常用的数组遍历:

|  |
| --- |
| var arr = [1,2,0,3,9];  for ( var i = 0; i <arr.length; i++){  console.log(arr[i]);  } |

### **优化版for循环**

使用变量，将长度缓存起来，避免重复获取长度，数组很大时优化效果明显

|  |
| --- |
| for(var j = 0,len = arr.length; j < len; j++){  console.log(arr[j]);  } |

### **for-in遍历**

for-in是为遍历对象而设计的，不适用于遍历数组。

遍历数组的缺点：数组的下标index值是数字，for-in遍历的index值"0","1","2"等是字符串。