Web开发技术-JavaScript

4 引用类型

内容提要

- 1 基本概念
  - 语法
  - 变量
  - 数据类型
    - 基本数据类型
    - 引用类型
  - 操作符和表达式
  - 语句
  - 函数
- 2 深入变量
- 3 作用域和内存问题
- 4 引用类型
  - Function 类型
  - Array 类型
  - 正则表达式和 RegExp 类型
  - 单体内置对象
- 5 面向对象的程序设计
- 6 函数表达式

4 引用类型

- 4.1 基本 JavaScript 类型
  - Function：函数
  - Array：数组
  - RegExp：正则表达式
  - Date：处理日期和时间
- 4.2 单体内置对象
  - Global：全局对象
  - Math：提供计算功能

4.1.1 Function 类型

- **函数的本质是对象**
  - 函数名实际上是指向函数对象的指针，不会与某个函数绑定。
    - 一个函数可以拥有多个名字
    - 函数对象的值（指针）可传递

- **定义函数**　　将函数对象的值（指针）存入变量 sum 中
  - 函数声明
  - 函数表达式**（不推荐）**

```
function sum (num1, num2) {
  return num1 + num2;
}
```

```
var sum = new function(num1, num2) {
  return num1 + num2;
}
```

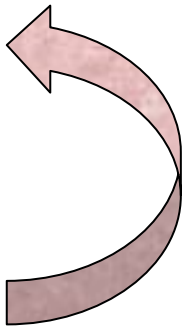
4.1.1 Function 类型

没有重载

```
function sum(num1, num2) {
  return num1 + num2;
}

function sum(num1, num2) {
  return num1 + num2 + 100;
}

var result = sum(100, 100);       // 300
```



创建第二个函数时
覆盖了引用第一个函数的变量 sum

4.1.1 Function 类型

- **arguments 对象：函数参数列表**
  - 函数内部以数组形式表示参数，可以通过 arguments 对象访问
  - 参数的命名只是提供便利，解析器不会验证命名参数
  - 通过 arguments.length 属性访问传入参数数量

```
function add() {
  if (arguments.length == 1) {
    alert(arguments[0] + 10);
  } else if (arguments.length == 2) {
    alert(arguments[0] + arguments[1]);
  }
}

add(20);      // 30
add(20, 30);      // 50
```

```
function howManyArgs() {
  alert(arguments.length);
}

howManyArgs("string", 45);      // 2
```

4.1.1 Function 类型

➢使用 arguments.callee 属性实现函数递归调用

```
/* 阶乘函数 */
function factorial(num) {
  if (num <= 1) {
    return 1;
  } else {
    return num * factorial(num - 1);
  }
}
```

```
function factorial(num) {
  if (num <= 1) {
    return 1;
  } else {
    return num * arguments.callee(num - 1);
  }
}
```

函数名不与函数对象绑定
递归调用时函数名与执行的函数紧耦合

4.1.2 Array 类型

- **数组对象**
  - 数组的每一项可以保存任何类型的数据
  - 数组大小自动动态调整，使用 `length` 属性查询数组长度

```
// 创建包含 "blue", "red", "green" 的数组
var colors = Array("blue", "red", "green");

var colors = ["blue", "red", "green"];

// 创建包含 3 项的空数组
var colors = Array(3);

// 创建包含 "blue", 1, true 的数组
var combined = Array("blue", 1, true);
```

```
// 访问数组长度
console.log(colors.length);  // 3
```

4.1.2 Array 类型

- Array类型
  - 使用 [从 0 开始的索引号] 读取或设置数组的值
  - length 可读可写。通过设置 length 属性，从数组末尾移除项或添加

```
var colors = Array("blue", "red", "green");

// 读取 colors 的第 1 项
console.log(colors[1]);          // "red"

// 设置 colors 的第 1 项为 "grey"
colors[1] = "grey";

// 删除 "red" 和 "green"
colors.length = 1;

// 在 colors 末尾添加 "black" 项
colors[colors.length] = "black";
```

4.1.2 Array 类型

- **Array.isArray(array) 检测数组**
  - 常与 if 搭配使用

```
var colors = Array("blue", "red");

if (Array.isArray(colors)) {
  alert("Yes, sir");
}
```

- **join() 方法拼接数组元素**

```
var colors = Array("blue", "red");

var string = colors.join("&");
console.log(string);   // "blue&red"
```

4.1.2 Array 类型

- **栈方法：push() / pop()**
  - push() 尾部推入；pop() 尾部取出

```
var colors = Array("blue", "red");

// 尾部推入 "grey" 和 "green"
colors.push("grey", "green");

alert(colors);
// blue, red, grey, green

// 尾部弹出 "green"
var item = colors.pop();
alert(item);          // green
```

- **队列方法：push() / shift() / unshift()**
  - push() 尾部推入；shift() 首部取出；unshift() 首部推入

```
var colors = Array("blue", "red");
// 尾部推入 "yellow"
colors.push("yellow");

// 首部取出 "blue"
var removed = colors.shift();
alert(removed);  // blue

// 首部推入 "purple", "pink"
colors.unshift("purple", "pink");
alert(colors);
// purple,pink,red,grey,yellow
```

4.1.2 Array 类型

- **reverse(), sort() 重排序**
  - reverse(): 逆序
  - sort(): 默认按**字符串**升序排列

Demo 2.6

```
var values = [0, 1, 5, 10, 15];

values.reverse();
alert(values);  // 15, 10, 5, 1, 0
```

```
var values = [0, 1, 5, 10, 15];

values.sort();
alert(values);  // 0, 1, 10, 15, 5
```

4.1.2 Array 类型

sort() 自定义排序

- sort(function)
  - sort() 可传入比较函数 function 作为参数
- function(value1, value2) 接受两个参数
  - 若排序后第一个参数应位于第二个参数之前，return -1；若相等， return 0；若应位于第二个参数后，return 1.

```
// compare 的简化写法
function easyCompare(value1, value2) {
  return value2 - value1;
}
```

Demo 2.7

```
var values = [1, 3, 5, 3, 1];
// 按大小降序排列
function compare(value1, value2) {
  if (value1 < value2) {
    return 1;
  } else if (value1 == value2) {
    return 0;
  } else {
    return -1;
  }
}

values.sort(compare);
alert(values);// 5, 3, 3, 1, 1
```

4.1.2 Array 类型

➢追加元素：concat(*items*)

- ➢创建当前数组的副本，再将接收到的参数添加到副本的末尾

Demo 2.8

```
var colors = Array("red", "green");

var newColors = colors.concat("black", ["white", "grey"]);

alert(newColors);
// red, green, black, white, grey
```

➢切片方法：slice(*start, end*)

- ➢返回数组从 start 到 end 项之间（除 end 项）所有元素的新数组

```
var colors = Array("blue", "red", "green");

var newColors = colors.slice(1, 3);
alert(newColors);
// red, green
```

4.1.2 Array 类型

- **替换方法：splice(*start, deleteNum, [items]*)**
  - 删除：指定 开始删除的项位置 start 和 删除项的数量 deleteNum
  - 插入：指定 起始位置 start，删除项数量 0 和 新插入的项 items
  - 替换：指定 起始位置 start，删除项数量 deleteNum 和新插入的项 items

Demo 2.9

```
var colors = Array("red", "green", "blue");
// 删除 green
colors.splice(1, 1);
alert(colors);          // red, blue

// 在位置 1 插入 pink, yellow
colors.splice(1, 0, "pink", "yellow");
alert(colors);          // red, pink, yellow, blue

// 将 pink 替换为 white
colors.splice(1, 1, "white");
alert(colors);          // red, white, yellow, blue
```

4.1.2 Array 类型

- **位置方法：indexOf(*item, [start]*) 和 lastIndexOf(*item, [start]*)**
  - item：要查找的项；start：开始查找的项的位置
  - 返回查找项的位置。若未找到，返回 -1.
  - indexOf() 从左开始查找；lastIndexOf() 从右开始查找

```
var colors = Array("red", "green", "blue");
// 查找 green
var num = colors.indexOf("green");
alert(num);     // 1

// 查找 hello
var num = colors.indexOf("hello");
alert(num);     // -1
```

4.1.2 Array 类型

- **迭代方法：查询数组中的项是否满足某个条件，迭代方法不会修改原数组中包含的值**

Demo 2.10

  - every(*function*)：对数组的每一项运行给定函数，若函数对**每一项**都返回 true，则返回 true.
  - some(*function*)：对数组的每一项运行给定函数，若函数对**任意一项**返回 true，则返回 true.

  - filter(*function*)：对数组的每一项运行给定函数，返回 true 的项构成的新数组。
  - map(*function*)：对数组的每一项运行给定函数，返回每次函数调用结果组成的数组。

  - forEach(*function*)：对数组的每一项运行给定函数，没有返回值。

4.1.2 Array 类型

- **every(*function*)**：对数组的每一项运行给定函数，若函数对**每一项**都返回 true，则返回 true.
- **some(*function*)**：对数组的每一项运行给定函数，若函数对**任意一项**返回 true，则返回 true.

```
// every()
var everyResult = values.every(function(item, index, array) {
  return item > 2;
});
alert(everyResult);          // false

// some()
var someResult = values.some(function(item, index, array) {
  return item > 2;
});
alert(someResult);           // true
```

4.1.2 Array 类型

- **filter(*function*)**：对数组的每一项运行给定函数，返回 true 的项构成的新数组。
- **map(*function*)**：对数组的每一项运行给定函数，返回每次函数调用结果组成的数组。

```
// filter()
var filterResult = values.filter(function(item, index, array) {
  return item > 2;
});
alert(filterResult);      // 3, 4, 5

// map()
var mapResult = values.map(function(item, index, array) {
  return item > 2;
});
alert(mapResult);          // false,false,true,true,true
```

4.1.2 Array 类型

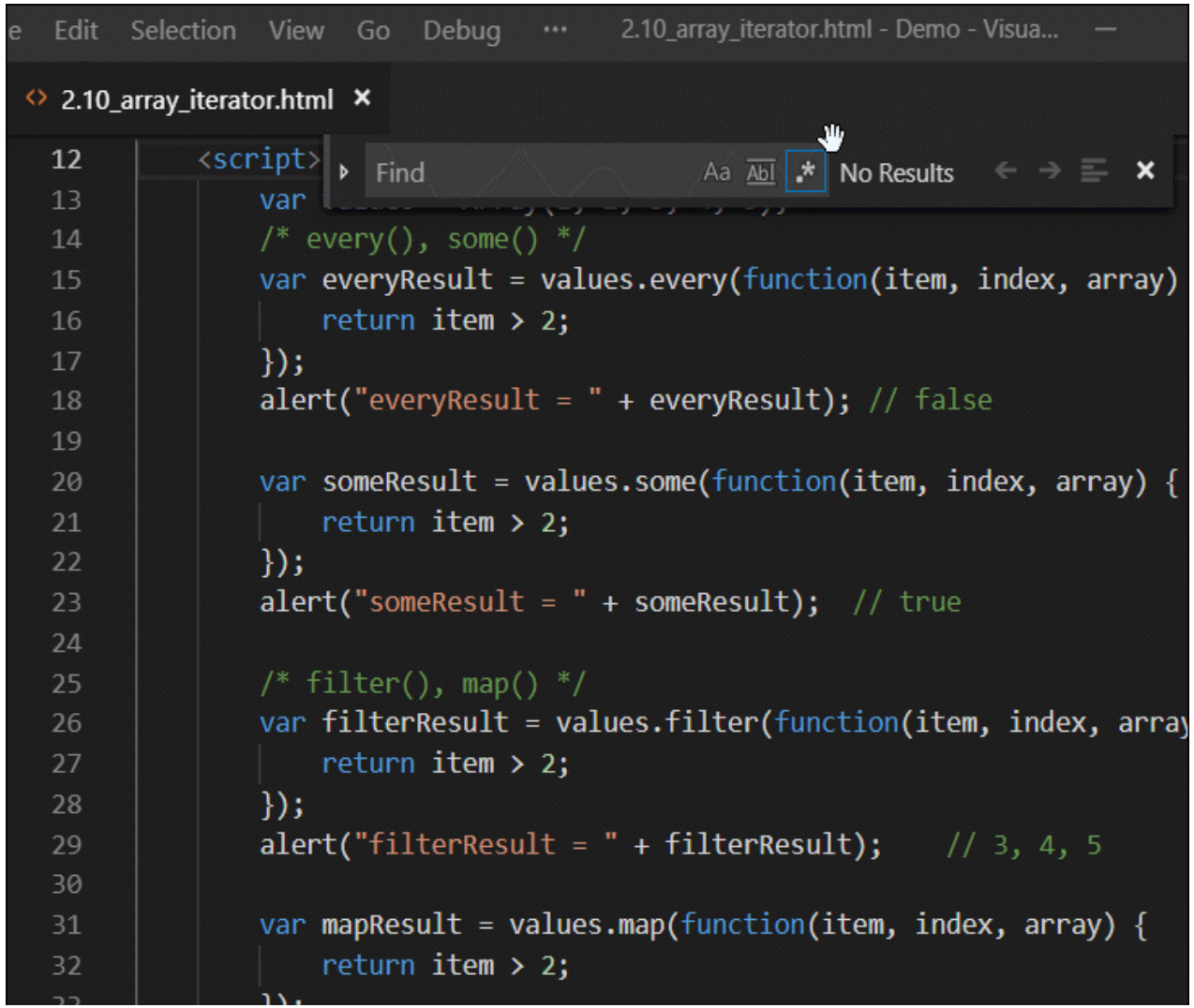
- **forEach(*function*)**：对数组的每一项运行给定函数，没有返回值

```
// forEach()
values.forEach(function(item, index, array) {
  console.log(item);          // 1 2 3 4 5
})
```

4.1.3 正则表达式

- **正则表达式（Regular Expression）**
  - 正则表达式使用单个字符串来描述、匹配一系列匹配某个句法规则的字符串。
  - 正则表达式通常被用来检索、替换那些匹配某个模式的文本。

- **例：使用正则表达式搜索以 Result 结尾的单词**



4.1.3 正则表达式

- **模式（Pattern）**
  - 一个正则表达式通常被称为一个模式（pattern），为用来描述或者匹配一系列匹配某个句法规则的字符串。
  - 例如：Handel、Händel 和 Haendel 这三个字符串，都可以由 H(a|ä|ae)ndel这个模式来描述。

4.1.3 正则表达式

- **元字符（Metacharacter）**
  - 模式中使用元字符代表某种状况下一个或多个字符。
  - 使用额外的 \ 转义元字符

常用元字符

代码	说明	举例：匹配 "hao123.com "
.	匹配除换行符以外的任意字符	.ao123\.com
\w	匹配字母或数字或下划线或汉字	hao\w23.com
\s	匹配任意的空白符	hao123.com\s
\d	匹配数字	hao\d23.com
\b	匹配单词的开始或结束	\bhao123.com
^	匹配字符串的开始	^hao123.com\s$
$	匹配字符串的结束	

4.1.3 正则表达式

重复

代码	说明	举例：匹配尾数为 1616 的 4 位以上数字
*	重复零次或更多次	\d*1616
+	重复一次或更多次	\d+1616
?	重复零次或一次	\d+16?6
{n}	重复 n 次	\d+(16){2}
{n,}	重复 n 次或更多次	
{n,m}	重复 n 到 m 次	

4.1.3 正则表达式

- **使用 [] 指定字符范围**
  - 如 [0-9] = \d;
  - 如 [a-zA-z0-9] = \w

例：匹配中国 11 位手机号?

```
1[2-9]\d{9}
```

- **使用 | 表示分枝条件**

例：匹配两种格式的座机号? 如
010-62336110, 3 位- 8 位
0411-84603562, 4 位 – 7 位

```
0\d{2}-\d{8}|0\d{3}-\d{7}
```

4.1.3 RegExp 类型

➢ **ECMAScript 使用 RegExp 类型支持正则表达式。需创建 RegExp 实例再执行匹配。**

创建正则表达式（RegExp 实例）语法：

```
var expression = / pattern / flags;
```

- pattern：模式，字符串
- flags：一个或多个标志，字符串

flags：标志位

标志	说明
g	*global：全局模式。*模式将被应用于所有字符串，否则在发现第一个匹配项时停止
i	*case-insensitive：不区分大小写模式。*确定匹配项时忽略模式与字符串的大小写
m	*multiline：多行模式。*在到达一行文本末尾时，继续查找下一行。

4.1.3 RegExp 类型

Demo 2.11

执行匹配：exec(*text*)

成功匹配，返回 [index, input] 数组；否则返回 null.

```
var text = "hi13031001616";
var pattern = /1[2-9]\d{9}/;
var execResult = pattern.exec(text);

alert(execResult.index + ", " + execResult.input);
// 2, hi13031001616
```

test(*text*)

成功匹配返回 true；否则返回 false

经常与 if 搭配使用

```
if (pattern.test(text)) {
    alert("Yes, sir!");
  } else {
    alert("Oh, no!")          // Yes, sir!
}
```

4.1.3 RegExp 类型

RegExp 构造函数属性

- 这些属性适用于作用域中的所有正则表达式，并且基于所执行的最近一次正则表达式操作而变化。

属性列表

属性名	说明
input	最近一次要匹配的字符串
lastMatch	最近一次的匹配项
lastParen	最近一次的捕获组
leftContent	input 字符串 lastMatch 之前的文本
rightContent	input 字符串 lastMatch 之后的文本
multiline	布尔值，是否使用多行模式（i）

```
var text = "hi13031001616";
var pattern = /1[2-9]\d{9}/;

if (pattern.test(text)) {
  alert(RegExp.input);
  // hi13031001616
  alert(RegExp.lastMatch);
  // 13031001616
}
```

4.2 单体内置对象

- **内置对象**
  - "由 ECMAScript 实现提供的、不依赖于宿主环境的对象，这些对象在 ECMAScript 程序执行之前就已经存在了"。
  - Object/Array/Function 等都是内置对象。
- **单体内置对象**
  - 开发人员不必显式地实例化的内置对象。
  - 两个单体内置对象：Global 和 Math.

4.2.1 单体内置对象：Global 对象

- **所有在全局作用域中定义的属性和函数，都是 Global 对象的属性**
  - 例如 isNaN()、isFinite()、parseInt() 以及 parseFloat()
- **eval(*text*) 方法执行 JavaScript 代码**
  - 当解析器发现代码中调用 eval() 方法时，它会将传入的参数当作实际的 ECMAScript 语句来解析，然后把执行结果插入到原位置。
  - 常用于标识符拼接。

```
var form1_wang = "<input type='text'>Form2</input>";
var form2_wang = "<input type='text'>Form2</input>";

eval("console.log(" + "form1" + "_wang);");
```

4.2.2 单体内置对象：Math 对象

➢ 使用 Math 对象的属性和方法进行数学运算

属性

属性	说明
Math.E	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

方法

方法	说明
Math.random()	返回 ⩾ 0 且 ＜ 1 的数
Math.max()	返回数组中的最大值
Math.min()	返回数组中的最小值
Math.ceil()	执行向上舍入
Math.floor()	执行向下舍入
Math.round()	执行标准舍入