Web开发技术-JavaScript

ECMAScript基本概念、变量和作用域

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

标识符

- 第一个字符必须是一个字母、下划线（_）或一个美元符号（$）
- 其他字符可以是字母、下划线、美元符号或数字
- 驼峰命名法（camelCase）

numberInput
firstChild


不推荐

number_input
first_child

注释

```
// 单行注释
```

```
/*
 * 多行
 * 注释
 */
```

语句

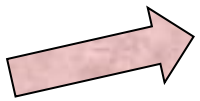
以分号";"结尾

语句和代码规范

- 以分号";"结尾
  - `var sum = a + b;`

- 代码块 { }

```
if (true) {
    alert("Hello World!");
}
```



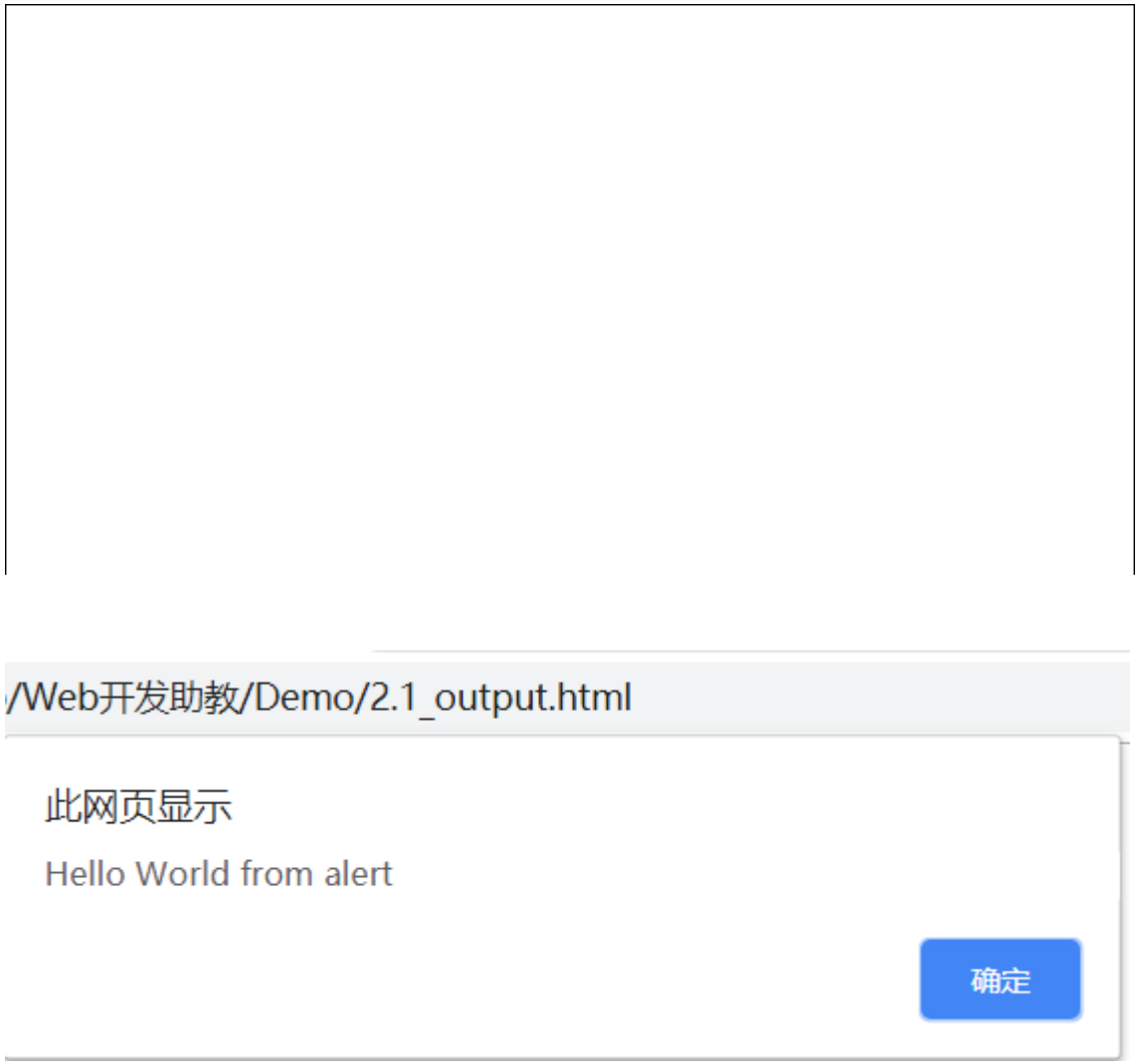
规范的代码书写：
不换行；
切即使代码块中只有一条语句，也要写花括号

在浏览器中查看 JavaScript 运行输出

Demo 2.1

- console.log(变量)
  - 在"检查"的 Console 窗口中查看
  - 例：
```
<script>
    console.log("Hello World!");
</script>
```

- alert(变量)
  - 自动弹窗输出结果
  - 例：
```
<script>
    alert("Hello World!");
</script>
```



变量声明

- 局部变量
  - var 操作符，后跟变量名
  - var message = "hello";
- 全局变量
  - 省略 var 操作符，直接声明变量名
  - message = "hello";
- 一条语句定义多个变量
  - var message = "hello", sender = “Mater", receiver = "BJFU";

数据类型

- 基本数据类型
  - Number（数值）：包括整数和浮点数
  - Boolean（布尔值）：取值为 true/false
  - String（字符串）：用单引号或双引号括起来的零个或多个单一的字符所组成
  - Undefined（未定义）：未定义或未初始化的变量值
  - Null（空值）：空对象指针

- 引用数据类型（又称复杂数据类型）
  - Object：在内存中的对象

松散类型（动态变量）

- 每个变量可以以保存任何类型的数据基本数据类型
  - `var message = "hi";  // String`
  - `message = 100;  // Number`

使用 typeof 操作符查看变量数据类型

那我怎么知道这个变量数据类型是什么?

- 用法
  - typeof 变量名
- 返回值
  - "number": 数值
  - "string": 字符串
  - "boolean": 布尔值
  - "undefined": 未定义
  - "object": 对象或 null
  - "function": 函数

```
var message = "Hello";
alert(typeof message);   // String
alert(typeof 100);    // Number
var nothing;
alert(typeof nothing);   // Undefined
```

数据类型转换

- 数据类型自动转换
  - 如果表达式中用（+）运算符，且其中一个操作数为字符串，另一个操作数为数值时，JavaScript 自动将数值转成字符串。

```
var x = "我今年" + 18;
var x = "15" + 8;
var y = 15 + 8;
```

结果：x = "我今年18"
结果：x = "158"
结果：y = 23

  - 如果表达式中用了其它运算符，JavaScript 自动将字符串转成数值。

```
var x = "30" / 5;
var x = "30" – 5;
```

结果：x = 6
结果：x = 25

1.3.1 基本数据类型：Number 类型

NaN

- NaN(Not a Number)
  - 特殊数值，表示一个本来要返回数值的操作数未返回数值的情况

- 任何涉及 NaN 的操作都会返回 NaN
  - alert(NaN / 5);　　// NaN

- 判断是否为 NaN：isNaN()
  - alert(isNaN(NaN));　　// true
  - alert((isNaN(10));　　// false

1.3.1 基本数据类型：Number 类型

- **数值转换：将其他类型值转为 Number 类型**
  - Number()
    - Boolean -> 1/0
    - Null -> 0
    - Undefined -> NaN
    - String ->
- parseInt()
- parseFloat()

解析字符串 ->
忽略字符串前空格

只包含数字：Number
空：0
其他格式：NaN

空：NaN
第一个字符是数字：直至非数字

```
alert(Number("   5"));  // 5
alert(Number("0005"));  // 5
alert(Number(""));   // 0
alert(Number("hello")); // NaN
alert(Number(true)); // 1
```

```
alert(parseInt("   1234blue")); // 1234
alert(parseInt("1234.11"));     // 1234
                                // . 不是数字
```

1.3.2 基本数据类型：String 类型

字符字面量（转义序列）

字符	意义	字符	意义
\b	后退一格(Backspace)	\t	制表(Tab)
\f	换页(Form feed)	\’	单引号
\n	换行(New line)	\"	双引号
\r	返回(Carriage return)	\\	反斜线(Backslash)

length 属性

- 任何字符串的长度可以访问其 length 属性获得

```
var text = "Hello!";
alert(text.length);  // 6
```

1.3.2 基本数据类型：String 类型

转换为字符串

- toString()
  - Number/Boolean/Object/String 值都具有 toString() 方法，Null/Undefined 类型值不能使用

```
var price = 5;
alert(a.toString()); // "5"
var yes = true;
alert(yes.toString());  // "true"
```

- String()
  - 值具有 toString() 方法 -> 调用 toString()
  - Undefined -> "undefined"
  - Null -> "null"

```
var price;
alert(String(price));  // "undefined"
var yes = null;
alert(String(yes));    // "null"
```

1.3.3 引用数据类型：Object 类型

引用类型

- 引用类型描述一类对象所具有的属性和方法。
- 是一种数据结构，用于将数据和功能组织在一起。

- 创建 Object 实例的方法：
  - new Object()
  - 对象字面量

1.3.3 引用数据类型：Object 类型

创建 Object 实例的两种方法

- new Object() 构造函数：

```
var person = new Object();
person.name = "Shaorong Wang";
person.corporation = "BJFU";
```

- 字面表达量：

更简洁，更流行

```
var person = {
  name: "Shaorong Wang",
  corporation: "BJFU"
};
```

实例/对象/属性

- new Object() 创建了 Object 引用类型的一个**实例**，并把该实例保存在变量 person 中。此时，person 也是一个**对象**。
- name/corporation 是 person 的两个**属性**。

1.3.3 引用数据类型：Object 类型

访问对象属性

- 点表示法：

```
alert(person.corporation); // BJFU
```

面向对象通用语法，较常用

- 方括号表示法：

```
alert(person["corporation"]); // BJFU
```

可以通过**变量**来访问**属性**

Demo 2.3

(for-in 遍历对象属性)

```
for (var prop in person) {
  console.log(prop + ": " + person[prop]);
}
```

```
name: Shaorong Wang
corporation: BJFU
```

1.4.1 操作符

赋值运算符

运算符	意义	运算符	意义
=	x = 5	/=	x = x / y
+=	x = x + y	%=	求余赋值
-=	x = x - y	*=	x = x * y

1.4.1 操作符

关系操作符

操作符	描述	举例
a >= b	如果 a 大于或者等于 b，返回 true	tries >= 2
a > b	如果 a 大于 b，返回 true	flag > 20
a <= b	如果 a 小于或等于 b，返回 true	i <= 0
a < b	如果 a 小巧或等于 b，返回 true	tries < 10

1.4.1 操作符

相等操作符

操作符	描述	举例	结果
a == b	相等和不相等。两个操作符先强制转型，再比较相等性。	"hello" == "hello"	true
		null == undefined	true
a != b		false == 0	true
		"5" == 5	true
a === b	全等和不全等。不经强制转型比较相等性。	"hello" === "hello"	true
		null === undefined	false
a !== b		false === 0	false
		"5" === 5	false

1.4.1 操作符

算数运算符

运算符	意义	运算符	意义	运算符	意义
+	加 (Addition)	/	除 (Division)	--	递减 (Decrement)
-	减 (Subtraction)	%	求余 (Modulus)	-	取负值 (Unary Negation)
*	乘(Multiplication)	++	递增(Increment)		

1.4.1 操作符

- **布尔操作符**
  - a&&b
    - 逻辑与(Logical AND)，若 a，b 都是 true，则结果为 true.
    - 短路操作：如果第一个操作数能决定结果，就不会对第二个操作数求值。
  - a||b
    - 逻辑或(Logical OR) ，若 a，b 任一是 true，则结果为 true.
  - !a
    - 逻辑非(Logical NOT) ，若 a 是 true，则结果为 false.

1.4.2 表达式

三元表达式

- 格式
  - 条件表达式? 值1: 值2;
- 如果条件表达式的结果是 true，返回 值 1，否则就返回 值 2.

```
var isTrue = true;
var yesOrNo = isTrue? "yes": "no";
alert(yesOrNo);      // "yes"
```

1.5 语句

if 语句

```
if (逻辑表达式) {
    语句块 1
} else {
    语句块 2
}
```

```
if (逻辑表达式 1) {
    语句块 1
} else if (逻辑表达式 2) {
    语句块 2
} else {
    语句块 3
}
```

1.5 语句

do-while 语句

```
do {
   statement
} while (expression);
```

```
var i = 0;
do {
   i += 2;
} while (i < 10);

alert(i);     // 10
```

1.5 语句

while 语句

```
while (expression) {
  statement
}
```

```
var i = 0;
while (i < 10) {
  i += 2;
}

console.log(i);     // 10
```

1.5 语句

- **for 语句**

```
for (初始值; 条件; 增量) {
  语句块
}
```

```
for (var i = 0; i < 10; i++) {
  console.log(i);
}
```

1.5 语句

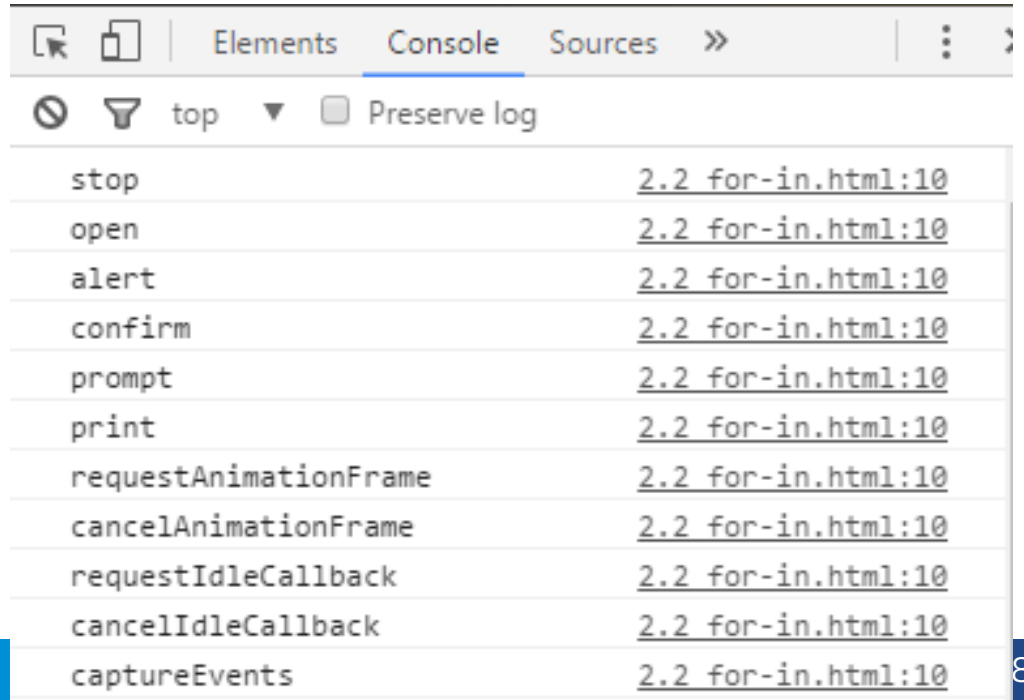
- **for-in 语句**
  - **更精准的迭代语句，用来枚举对象的属性**

```
for (属性 in 对象) {
  语句块
}
```

```
for (var propName in window) {
  console.log(propName);
}
```

Demo 2.2

```
<script>
for (var propName in window) {
  console.log(propName);
}
</script>
```



1.5 语句

- **break 语句**
  - **立刻退出循环并强制继续执行循环后面的语句**

```
var num = 0;
for (var i = 0; i < 10; i++) {
  if (i % 5 == 0) {
    break;
  }
  num++;
}
alert(num);      // 4
```

- **continue 语句**
  - **立即退出循环，退出后继续从循环的顶部执行**

```
var num = 0;
for (var i = 0; i < 10; i++) {
  if (i % 5 == 0) {
    continue;
  }
  num++;
}
alert(num);      // 8
```

1.5 语句

label 语句

用于在代码中添加标签，以便将来使用。

经常与 break/continue 和循环嵌套配合使用。

```
var num = 0;
outermost:   // 用 outermost 表示外部循环
for (var i = 0; i < 10; i++) {
  for (var j = 0; j < 10; j++) {
    if (i == 5 && j == 5) {
      break outermost;  // 直接跳出外层循环
    }
    num++;
  }
}
console.log(num);      // 55
```

1.6 函数

使用 function 关键字声明

```
function functionName(arg0, arg1,…argN) {
  statements
}
```

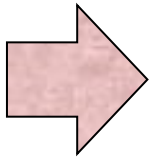
```
// 函数声明
function sayHi(name, message) {
  alert(alert("Hello " + name + ", " + message);
}

// 函数调用
sayHi("Wang", "how are you?");      // Hello Wang, how are you?
```

2 深入变量

- **基本类型和引用类型的*值***
  - 基本类型值：简单数据段，**按值访问。存储*实际值*。**
  - 引用类型值：多个值构成的对象，**按引用访问。**
    - **存储在变量中的是一个指向存储在堆中对象的*指针*。**

- **基本类型的值复制**
- 如果从一个变量向另一个变量复制基本类型的值，会在变量对象上创建一个新值，然后把该值复制到为新变量分配的位置上。

```
var num1 = 5;
var num2 = num1;
```



复制前的变量对象

num1	5 （Number 类型）

复制后的变量对象

num2	5 （Number 类型）
num1	5 （Number 类型）

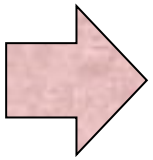
2 深入变量

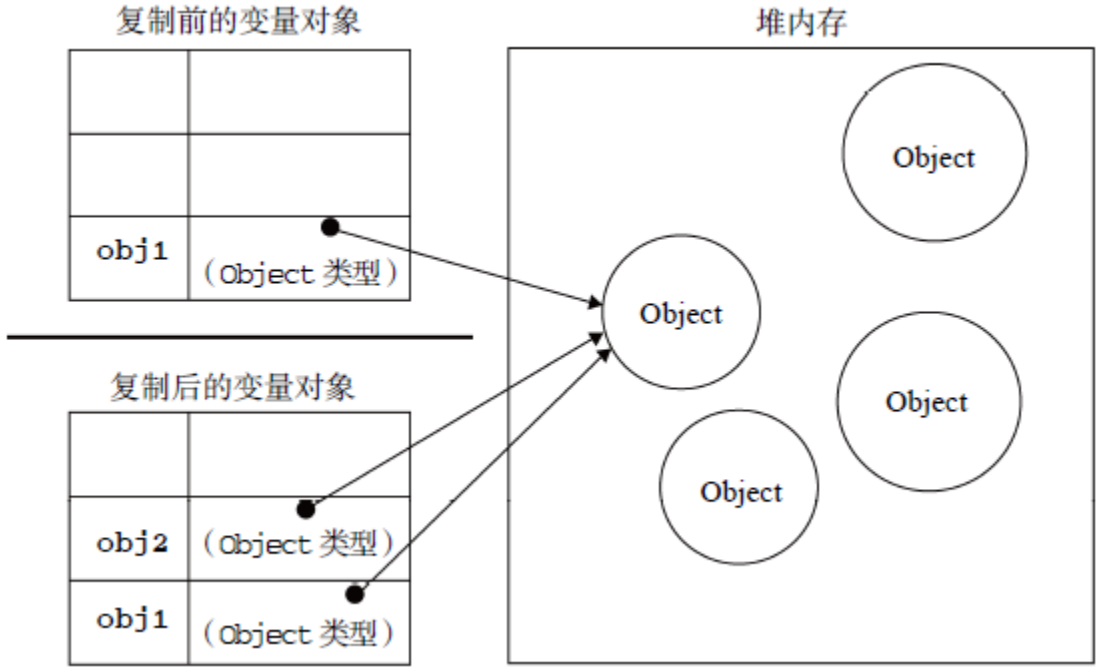
引用类型的值复制

当从一个变量向另一个变量复制引用类型的值时，会将存储在变量对象中的**指针**复制一份放到为新变量分配的空间中。

复制操作结束后，两个变量实际上将引用**同一个对象**。

```
var obj1 = new Object();
var obj2 = obj1;
obj1.name = "Wang";
alert(obj2.name); //"Wang"
```





2 深入变量

➢向函数传递参数

➢ECMAScript 中所有函数的参数都是**按值传递**的。即把函数外部的值**复制**给函数内部的参数。

➢基本数据类型：传递**实际值**的副本

➢引用数据类型：传递**指针**的副本

Demo 2.4

```
function setName(obj) {
  obj.name = "Jia";
}

var person = new Object();
person.name = "Wang";
setName(person);
alert(person.name);        // Jia
```

3 作用域和内存问题

没有块级作用域

- 代码块内的变量在执行后不会被销毁。

```
if (true) {
  var message = "hello";
}
alert(message); // "hello"
```

垃圾收集

- JavaScript 具有自动垃圾收集机制：自动找出不再使用的变量，释放其占用的内存。
- **若不再需要变量，将其值设为 null.**

```
message = null;
```