JavaScript

JavaScript（简称"JS"）是当前最流行、应用最广泛的客户端脚本语言，用来在网页中添加一些动态效果与交互功能，在 Web 开发领域有着举足轻重的地位。

JavaScript 是一种轻量级的编程语言。JavaScript 是可插入 HTML 页面的编程代码。 JavaScript 插入 HTML 页面后，用来给网页添加动态功能。可由所有的现代浏览器来解析执行 JavaScript 代码。JavaScript 本来应该叫 Livescript，但是在发布前夕，临时把名字改为了 JavaScript，JavaScript 跟 java 没有关系。

JavaScript 与 HTML 和 CSS 共同构成了我们所看到的网页，其中：

- HTML 用来定义网页的内容，例如标题、正文、图像等；
- CSS 用来控制网页的外观，例如颜色、字体、背景等；
- JavaScript 用来实时更新网页中的内容，例如从服务器获取数据并更新到网页中，修改某些标签的样式或其中的内容等，可以让网页更加生动。

JavaScript程序

JavaScript程序不能够独立运行，只能在宿主环境中执行。一般情况下可以把 JavaScript 代码放在网页中，借助浏览器环境来运行。

JavaScript 的书写位置

JavaScript 是一门弱类型动态的脚本语言。在页面中使用JavaScript有三种方式：

1、行内javascript：

在元素的事件属性中书写，由于写在html中，导致代码冗余，因此不常用。

如：onclick属性等

```
<h1>行内 JavaScript 效果图</h1>
<p>
请您单击按钮测试!
</p>
<button type="button" onclick="alert('欢迎来到国家信息安全水平考试,nisp 系列课程!')">NISP</button>
```

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>NISP</title>
</head>

<body>

    <h1>行内JavaScript效果图</h1>
    <p>
```

```
        请您单击按钮测试!
    </p>
    <button type="button" onclick="alert('欢迎来到国家信息安全水平考试,nisp系列课
程!')">NISP</button>

</body>

</html>
```

2、内嵌式：

可以书写body标签内部的一对script标签中（学习中常用）。

script标签有一个属性type,属性值是“text/javascript”表示书写的是纯文本的javascript语言。

```
<body>
<script >
alert('欢迎来到国家信息安全水平考试,nisp 系列课程!')
</script>
</body>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    <script >
        alert('欢迎来到国家信息安全水平考试,nisp系列课程!')
    </script>
</body>
</html>
```

3、外链式：

通过外部引入js文件，将引入文件书写head标签内部（工作中常用），通过 src 属性引入文件地址。

JavaScript 文件扩展名是.js，JavaScript 也可以简称 js

js文件的书写：直接在js文件中书写我们想显示的内容。

js脚本语言不能直接在浏览器中加载，必须依托html载体实现。

```
<script type="text/javascript" src="js/外联.js">
</script>
```

html 页面中

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    <script type="text/javascript" src="js/外联.js">
    </script>
</body>
</html>
```

外联js

```
alert('欢迎来到国家信息安全水平考试,nisp系列课程! ')
```

JavaScript 代码执行顺序

浏览器在解析 HTML 文档时，将根据文档流从上到下逐行解析和显示。JavaScript 代码也是 HTML 文档的组成部分，因此 JavaScript 脚本的执行顺序也是根据 标签的位置来确定的。

使用浏览器测试下面示例，会看到 JavaScript 代码从上到下逐步被解析的过程。

```
<!DOCTYPE html>
<script>alert("顶部脚本");</script>
<html>

<head>
    <meta charset="UTF-8">
    <title>test</title>
    <script>alert("头部脚本");</script>
</head>

<body>
    <h1>网页标题</h1>
    <script>alert("页面脚本");</script>
    <p>正文内容</p>
</body>
<script>alert("底部脚本");</script>

</html>
```

在浏览器中浏览上面示例网页，首先弹出提示文本“顶部脚本”，然后显示网页标题“test”，接着弹出提示文本“头部脚本”，下面才显示一级标题文本“网页标题”，继续弹出提示文本“页面脚本”， 接着显示段落文本“正文内容”，最后弹出提示文本“底部脚本”。

对于导入的 JavaScript 文件，也将按照 < script> 标签在文档中出现的顺序来执行，而且执行过程是文档解析的一部分，不会单独解析或者延期执行。

JavaScript 注释

单行注释：

只能注释单行文本，如果换行将报错。

如：// 单行注释

快捷键：ctrl + /

多行注释：

可以一次注释多行文本

如：/*

多行注释，

可以一次注释多行文本

*/

快键键：ALT +shift + A

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
    // 单行注释，只能让当前行失效
    // alert(1);
    // 多行注释
    /* alert(2);
    alert(3); */

    /**
     * 多行注释常用于书写一些说明文档
     *
     * ***/
    // 单行注释常用于注释代码，取消某一行很容易
    // alert(2);
    // alert(3)

    </script>
</body>
</html>
```

JavaScript中的几个重要概念

我们讲解一下 JavaScript中的几个简单的概念，包括标识符、关键字、保留字、大小写和字面量。这些基本概念虽然不能直接提升我们的编程能力，但它们是 JavaScript 的基本组成元素。

1. 标识符

所谓标识符（Identifier），就是名字。JavaScript 中的标识符包括变量名、函数名、参数名、属性名、类名等。

合法的标识符应该注意以下强制规则：

- 第一个字符必须是字母、下划线（_）或美元符号（$）。
- 除了第一个字符外，其他位置可以使用 Unicode 字符。一般建议仅使用 ASCII 编码的字母，不建议使用双字节的字符。
- 不能与 JavaScript 关键字、保留字重名。
- 可以使用 Unicode 转义序列。例如，字符 a 可以使用“\u0061”表示。

示例：在下面示例中，str 就是变量的名字。

```
var str = "nisp";
document.write(str);
var str1 = "\u0061";
document.write(str1);
```

第1行代码定义了一个变量，名字为 str，第2行通过 str 这个名字使用了变量。

2. 关键字

关键字（Keyword）就是 JavaScript 语言内部使用的一组名字（或称为命令）。这些名字具有特定的用途，用户不能自定义同名的标识符，具体说明如表所示。

break	**delete**	**if**	**this**	**while**
case	do	in	throw	with
catch	else	instanceof	try	
continue	finally	new	typeof	
debugger（ECMAScript 5 新增）	for	return	var	
default	function	switch	void	

3. 保留字

保留字就是 JavaScript 语言内部预备使用的一组名字（或称为命令）。这些名字目前还没有具体的用途，是为 JavaScript 升级版本预留备用的，建议用户不要使用。具体说明如表所示。

abstract	**double**	**goto**	**native**	**static**
boolean	enum	implements	package	super
byte	export	import	private	synchronized

abstract	double	goto	native	static
char	extends	int	protected	throws
class	final	interface	public	transient
const	float	long	short	volatile

ECMAScript 3 将 Java 所有关键字都列为保留字，而 ECMAScript 5 规定较为灵活，例如：

- 在非严格模式下，仅规定 class、const、enums、export、extends、import、super 为保留字，其他 ECMAScript 3 保留字可以自由使用；
- 在严格模式下，ECMAScript 5 变得更加谨慎，严格限制 implements、interface、let、package、private、protected、public、static、yield、eval（非保留字）、arguments（非保留字）的使用。

JavaScript 预定义了很多全局变量和函数，用户也应该避免使用它们，具体说明如表所示。

arguments	encodeURL	Infinity	Number	RegExp
Array	encodeURLComponent	isFinite	Object	String
Boolean	Error	isNaN	parseFloat	SyntaxError
Date	eval	JSON	parseInt	TypeError
decodeURL	EvalError	Math	RangeError	undefined
decodeURLComponent	Function	NaN	ReferenceError	URLError

不同的 JavaScript 运行环境都会预定义一些全局变量和函数，上表列出的仅针对 Web 浏览器运行环境。

无论是在严格模式下还是在非严格模式下，都不要在定义变量名、函数名或者属性名时使用上面列举出的保留字，以免同学们入坑。

4. 区分大小写

JavaScript 严格区分大小写，所以 Hello 和 hello 是两个不同的标识符。

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>JavaScript</title>
</head>

<body>
    <p>这是一个JavaScript程序</p>
    <script type="text/javascript">
        var hello = "123";
        document.write(hello);
        var Hello = "456";
        document.write(hello);
    </script>
</body>
```

```
</html>
```

为了避免输入混乱和语法错误，建议采用小写字符编写代码，在以下特殊情况下可以使用大写形式：

大驼峰命名法规定每个单词的首字母都大写，并且没有下划线分隔。这种命名方式主要用于类名的命名，例如：`Person、Car、Dog`等。大驼峰命名使类名在代码中更容易与函数和变量区分开来，增加了可读性。

小驼峰命名法则规定第一个单词的首字母小写，后面每个单词的首字母大写，同样没有下划线分隔。这种命名方式主要用于变量和函数的命名，例如：`firstName、getMaxValue、calculateAge`等。小驼峰命名方式使变量和函数名的可读性更好，同时也提高了代码的可维护性。

下划线命名法：下划线命名法也称为蛇形命名法，它使用下划线来分隔单词。例如：`first_name、get_max_value`。

构造函数的首字母建议大写。构造函数不同于普通函数。

示例：下面示例调用预定义的构造函数 Date()，创建一个时间对象，然后把时间对象转换为字符串显示出来。

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>JavaScript</title>
</head>

<body>
    <p>这是一个JavaScript程序</p>
    <script type="text/javascript">
    d = new Date();  //获取当前日期和时间
    document.write(d);  // 显示日期
    </script>
</body>

</html>
```

提示：上述都是约定俗成的一般习惯，不构成强制性要求，用户可以根据个人习惯进行命名。

5. 直接量

字面量（Literal）也叫直接量，就是具体的值，即能够直接参与运算或显示的值，如字符串、数值、布尔值、正则表达式、对象直接量、数组直接量、函数直接量等。

示例：下面示例分别定义不同类型的直接量：字符串、数值、布尔值、正则表达式、特殊值、对象、数组和函数。

```
//空字符串直接量
1  //数值直接量
true  //布尔值直接量
/a/g  //正则表达式直接量
null  //特殊值直接量
{}  //空对象直接量
[]  //空数组直接量
function(){}  //空函数直接量，也就是函数表达式
```

JS变量定义和赋值

变量是所有编程语言的基础之一，可以用来存储数据，例如字符串、数字、布尔值、数组等，并在需要时设置、更新或者读取变量中的内容。我们可以将变量看作一个值的符号名称。

1. 变量的命名规则

在 JavaScript 中，变量名称并不能随便定义，需要遵循标识符的命名规则，如下所示：

- 变量名中可以包含数字、字母、下划线`_`、美元符号`$`；
- 变量名中不能出现汉字；
- 变量名中不能包含空格；
- 变量名不能是 JavaScript 中的关键字、保留字；
- 变量名不能以数字开头，即第一个字符不能为数字。

在定义变量时，变量名要尽量有意义，让自己或者他人能轻易看懂，例如可以使用 name 来定义一个存储姓名的变量、使用 dataArr 来定义一个数组类型的变量。

当变量名中包含多个英文单词时，推荐使用驼峰命名法（大驼峰：每个单词首字母大写，例如 FileType、DataArr；小驼峰：第一个单词首字母小写后面的单词首字母大写，例如 fileType、dataArr）。

提示：JavaScript 中的关键字、保留字不能作为变量的名称。

2. 定义变量

在 JavaScript 中，定义变量需要使用`var`关键字，语法格式如下：

var 变量名;

举几个例子：

```
var str;  //用来存储字符串
var age;  //用来存储年龄
var name;  //用来存储姓名
```

定义变量时，可以一次定义一个或多个变量，若定义多个变量，则需要在变量名之间使用逗号`,`分隔开，如下例所示：

var a, b, c; // 同时声明多个变量

变量定义后，如果没有为变量赋值，那么这些变量会被赋予一个初始值——undefined（未定义）。

3. 为变量赋值

变量定义后，可以使用等于号`=`来为变量赋值，等号左边的为变量的名称，等号右边为要赋予变量的值，如下例所示：

```
var num;    // 定义一个变量 num
num = 1;    // 将变量 num 赋值为 1
```

此外，也可以在定义变量的同时为变量赋值，如下例所示：

```
var num = 1;                // 定义一个变量 num 并将其赋值为 1
var a = 2, b = 3, c = 4;    // 同时定义 a、b、c 三个变量并分别赋值为 2、3、4
var a = 2,
b = 3,
c = 4;                      // 为了让代码看起来更工整，上一行代码也可以写成这样
```

4. let 和 const 关键字

2015 年以前，JavaScript 只能通过 var 关键字来声明变量，在 ECMAScript6（ES6）发布之后，新增了 let 和 const 两个关键字来声明变量，其中：

- 使用 let 关键字声明的变量只在其所在的代码块中有效（类似于局部变量），并且在这个代码块中，同名的变量不能重复声明；
- const 关键字的功能和 let 相同，但使用 const 关键字声明的变量还具备另外一个特点，那就是 const 关键字定义的变量，一旦定义，就不能修改（即使用 const 关键字定义的为常量）。

注意：IE10 及以下的版本不支持 let 和 const 关键字。

示例代码如下：

```
let name = "小明";      // 声明一个变量 name 并赋值为“小明”
let age  = 11;          // 声明一个变量 age
let age  = 13;          // 报错：变量 age 不能重复定义
const PI = 3.1415       // 声明一个常量 PI，并赋值为 3.1415
console.log(PI)         // 在控制台打印 PI
```

JS数据类型（基本数据类型+引用类型）

数据类型指的是可以在程序中存储和操作的值的类型，每种编程语言都有其支持的数据类型，不同的数据类型用来存储不同的数据，例如文本、数值、图像等。

JavaScript 是一种动态类型的语言，在定义变量时不需要提前指定变量的类型，变量的类型是在程序运行过程中由 JavaScript 引擎动态决定的，另外，您可以使用同一个变量来存储不同类型的数据，例如：

```
var a;  // 此时 a 为 Undefined
a = "https://cisp.cn/"; // 此时 a 为 String 类型
a = 123;  // 此时 a 为 Number 类型
```

JavaScript 中的数据类型可以分为两种类型：

- 基本数据类型（值类型）：字符串（String）、数字（Number）、布尔（Boolean）、空（Null）、未定义（Undefined）、Symbol；

- 引用数据类型：对象（Object）、数组（Array）、函数（Function）。

提示：Symbol 是 ECMAScript6 中引入的一种新的数据类型，表示独一无二的值。

typeof 操作符

在开始介绍各种数据类型之前，先来了解一下 typeof 操作符，使用 typeof 操作符可以返回变量的数据类型。

typeof 操作符有带括号和不带括号两种用法，如下例所示：

```
typeof x;       // 获取变量 x 的数据类型
typeof(x);      // 获取变量 x 的数据类型
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    <script>
var a;
var b = 1;
var c = 'nisp';
var d = 2>3;
var e = null;
var f = function() {alert(nisp)};
var g = [1,2,3];
var h = {a:1,b:2};
console.log(typeof a)
console.log(typeof b)
console.log(typeof c)
console.log(typeof d)
console.log(typeof e)
console.log(typeof f)
console.log(typeof g)
console.log(typeof h)
</script>
</body>
</html>
```

1. JS 基本数据类型

1) String 类型

字符串（String）类型是一段以单引号`''`或双引号`""`包裹起来的文本，例如 '123'、"abc"。需要注意的是，单引号和双引号是定义字符串的不同方式，并不是字符串的一部分。

定义字符串时，如果字符串中包含引号，可以使用反斜杠`\`来转义字符串中的引号，或者选择与字符串中不同的引号来定义字符串，如下例所示：

```
var str = "Let's have a cup of coffee.";  // 双引号中包含单引号
var str = 'He said "Hello" and left.';    // 单引号中包含双引号
var str = 'We\'ll never give up.';        // 使用反斜杠转义字符串中的单引号
```

2) Number 类型

数值（Number）类型用来定义数值，JavaScript 中不区分整数和小数（浮点数），统一使用 Number 类型表示，如下例所示：

```
var num1 = 123;     // 整数
var num2 = 3.14;    // 浮点数
```

注意：Number 类型所能定义的数值并不是无限的，JavaScript 中的 Number 类型只能表示 -(253 -1) 到 (253 -1) 之间的数值。

对于一些极大或者极小的数，也可以通过科学（指数）计数法来表示，如下例所示：

```
var y=123e5;      // 123 乘以 10 的 5 次方，即 12300000
var z=123e-5;     // 123 乘以 10 的 -5 次方，即 0.00123
```

另外，Number 类型中还有一些比较特殊的值，分别为 Infinity、-Infinity 和 NaN，其中

- Infinity：用来表示正无穷大的数值，一般指大于 1.7976931348623157e+308 的数；
- -Infinity：用来表示负无穷大的数值，一般指小于 5e-324 的数；
- NaN：即非数值（Not a Number 的缩写），用来表示无效或未定义的数学运算结构，例如 0 除以 0。

提示：如果某次计算的结果超出了 JavaScript 中 Number 类型的取值范围，那么这个数就会自动转化为无穷大，正数为 Infinity，负数为 -Infinity。

3) Boolean 类型

布尔（Boolean）类型只有两个值，true（真）或者 false（假），在做条件判断时使用的比较多，您除了可以直接使用 true 或 false 来定义布尔类型的变量外，还可以通过一些表达式来得到布尔类型的值，例如：

```
var a = true;   // 定义一个布尔值 true
var b = false;  // 定义一个布尔值 false
var c = 2 > 1;  // 表达式 2 > 1 成立，其结果为“真（true）”，所以 c 的值为布尔类型的 true
var d = 2 < 1;  // 表达式 2 < 1 不成立，其结果为“假（false）”，所以 c 的值为布尔类型的
false
```

4) Null 类型

Null 是一个只有一个值的特殊数据类型，表示一个“空”值，即不存在任何值，什么都没有，用来定义空对象指针。

使用 typeof 操作符来查看 Null 的类型，会发现 Null 的类型为 Object，说明 Null 其实使用属于 Object（对象）的一个特殊值。因此通过将变量赋值为 Null 我们可以创建一个空的对象。

5) Undefined 类型

Undefined 也是一个只有一个值的特殊数据类型，表示未定义。当我们声明一个变量但未给变量赋值时，这个变量的默认值就是 Undefined。例如：

```
var num;console.log(num);  // 输出 undefined
```

在使用 typeof 操作符查看未赋值的变量类型时，会发现它们的类型也是 undefined。对于未声明的变量，使用 typeof 操作符查看其类型会发现，未声明的变量也是 undefined，示例代码如下：

```
var message;console.log(typeof message);  // 输出 undefined
console.log(typeof name);     // 输出 undefined
```

6) Symbol 类型

Symbol 是 ECMAScript6 中引入的一种新的数据类型，表示独一无二的值，Symbol 类型的值需要使用 Symbol() 函数来生成，如下例所示：

```
var str = "123";
var sym1 = Symbol(str);
var sym2 = Symbol(str);
console.log(sym1);          // 输出 Symbol(123)
console.log(sym2);          // 输出 Symbol(123)
console.log(sym1 == sym2);  // 输出 false ：虽然 sym1 与 sym2 看起来是相同的，但实际上它们并不一样，根据 Symbol 类型的特点，sym1 和 sym2 都是独一无二的
```

2. JS 引用数据类型

1) Object 类型

JavaScript 中的对象（Object）类型是一组由键、值组成的无序集合，定义对象类型需要使用花括号`{ }`，语法格式如下：

{name1: value1, name2: value2, name3: value3, ..., nameN: valueN}

其中 name1、name2、name3、...、nameN 为对象中的键，value1、value2、value3、...、valueN 为对应的值。

在 JavaScript 中，对象类型的键都是字符串类型的，值则可以是任意数据类型。要获取对象中的某个值，可以使用`对象名.键`的形式，如下例所示：

```
var person = {
name: 'Bob',
age: 20,
tags: ['js', 'web', 'mobile'],
city: 'Beijing',
hasCar: true,
zipcode: null
};
console.log(person.name);       // 输出 Bob
console.log(person.age);        // 输出 20
```

2) Array 类型

数组（Array）是一组按顺序排列的数据的集合，数组中的每个值都称为元素，而且数组中可以包含任意类型的数据。在 JavaScript 中定义数组需要使用方括号`[ ]`，数组中的每个元素使用逗号进行分隔，例如：

```
[1, 2, 3, 'hello', true, null]
```

另外，也可以使用 Array() 函数来创建数组，如下例所示：

```
var arr = new Array(1, 2, 3, 4);
console.log(arr);       // 输出 [1, 2, 3, 4]
```

数组中的元素可以通过索引来访问。数组中的索引从 0 开始，并依次递增，也就是说数组第一个元素的索引为 0，第二个元素的索引为 1，第三个元素的索引为 2，以此类推。如下例所示：

```
var arr = [1, 2, 3.14, 'Hello', null, true];
console.log(arr[0]);  // 输出索引为 0 的元素，即 1
console.log(arr[5]);  // 输出索引为 5 的元素，即 true
console.log(arr[6]);  // 索引超出了范围，返回 undefined
```

3) Function 类型

函数（Function）是一段具有特定功能的代码块，函数并不会自动运行，需要通过函数名调用才能运行，如下例所示：

```
function sayHello(name){
return "Hello, " + name;
}
var res = sayHello("Peter");
console.log(res);  // 输出 Hello, Peter
```

字符串运算符

JavaScript 中的`+`和`+=`运算符除了可以进行数学运算外，还可以用来拼接字符串，其中：

- `+`运算符表示将运算符左右两侧的字符串拼接到一起；
- `+=`运算符表示先将字符串进行拼接，然后再将结果赋值给运算符左侧的变量。

示例代码如下：

```
var x = "Hello ";
var y = "World!";
var z = x + y;
console.log(z);  // 输出：Hello World!
x += y;
console.log(x);  // 输出：Hello World!
```

字符串

在js中，一切数据本质上都是对象，因此所有的数据都有相关的操作方法。

值类型的数据在创建的时候，也是通过对象方式创建的，只不过太简单了，因此有一个地址存储就可以了有一些数据实在太简单了，就不需要方法操作了。

concat()

表示字符串的拼接。参数也是很灵活，可以书写字面量，变量，散的值等。返回值是拼接后的字符串

split()

字符串转为数组。（数组中每一项仍是字符串），参数是切割的标志。不传递不会切割

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
<script>
// 字符串拼接
// var str = 'hello';
// // 不会影响原始字符串，返回值是拼接后的，参数中，其它类型数据会转成字符串进行拼接
// var result = str.concat('nisp', 'abc', true, 100);
// console.log(str, result);

// 切割字符串
var str = 'hello nisp, hello nisp';
// 原始字符串不受影响，返回值是切割后的数字
// console.log(str.split(' '), str);
// // 没有切割标志，整个字符串变成数组的一个成员
// console.log(str.split());
// // 空字符串，将每个字符单独切割出来
// console.log(str.split(''));
// 数组转字符串
// var result = str.split(' ')
// // 以什么切割，以什么拼接
// console.log(result.join(' '));
console.log(str.split(' ').join(' '));
</script>
</body>
</html>
```

JS函数（function）的定义和使用

函数是一组执行特定任务（具有特定功能）的，可以重复使用的代码块，前面几节中用到的 alert()、write() 就是 JavaScript 中内置的函数。

除了使用内置函数外，我们也可以自行创建函数（自定义函数），然后在需要的地方调用这个函数，这样不仅可以避免编写重复的代码，还有利于代码的后期维护。本节我们主要来介绍一下如何使用 JavaScript 编写一个自己的函数。

JS 定义函数

JS 函数声明需要以 function 关键字开头，之后为要创建的函数名称，function 关键字与函数名称之间使用空格分开，函数名之后为一个括号`( )`，括号中用来定义函数中要使用的参数（多个参数之间使用逗号`,`分隔开），一个函数最多可以有 255 个参数，最后为一个花括号`{ }`，花括号中用来定义函数的函数体（即实现函数的代码），如下所示：

```
function functionName(parameter_list) {
  // 函数中的代码
}
```

示例代码如下：

```
function sayHello(name){
document.write("Hello " + name);
}
```

上面示例中定义了一个函数 sayHello()，该函数需要接收一个参数 name，调用该函数会在页面中输出“Hello ...”。

JS 调用函数

一旦定义好了一个函数，我们就可以在当前文档的任意位置来调用它。调用函数非常简单，只需要函数名后面加上一个括号即可，例如 alert()、write()。注意，如果在定义函数时函数名后面的括号中指定了参数，那么在调用函数时也需要在括号中提供对应的参数。

示例代码如下：

```
function sayHello(name){
document.write("Hello " + name);
}
// 调用 sayHello() 函数
sayHello('nisp');
```

提示：JavaScript 对于大小写敏感，所以在定义函数时 function 关键字一定要使用小写，而且调用函数时必须使用与声明时相同的大小写来调用函数。

参数的默认值

在定义函数时，您可以为函数的参数设置一个默认值，这样当我们在调用这个函数时，如果没有提供参数，就会使用这个默认值作为参数值，如下例所示：

```
function sayHello(name = "World"){
document.write("Hello " + name);
}
sayHello();                 // 输出：Hello World
sayHello('nisp');           // 输出：Hello nisp
```

JS 函数返回值

在函数中可以使用 return 语句将一个值（函数的运行结果）返回给调用函数的程序，这个值可以是任何类型，例如数组、对象、字符串等。对于有返回值的函数，我们可以会使用一个变量来接收这个函数的返回值，示例代码如下：

```
function getSum(num1, num2){
return num1 + num2;
}
var sum1 = getSum(7, 12);      // 函数返回值为：19
var sum2 = getSum(-5, 33);     // 函数返回值为：28
```

提示：return 语句通常在函数的末尾定义，当函数运行到 return 语句时会立即停止运行，并返回到调用函数的地方继续执行。

另外，一个函数只能有一个返回值，若要返回多个值则，则可以将值放入一个数组中，然后返回这个数组即可，如下例所示：

```
function division(dividend, divisor){
var quotient = dividend / divisor;
var arr = [dividend, divisor, quotient] ;
return arr;
}
var res = division(100, 4);
document.write(res[0]);         // 输出：100
document.write(res[1]);         // 输出：4
document.write(res[2]);         // 输出：25
```

JS 函数表达式

函数表达式与声明变量非常相似，是另外一种声明函数的形式，语法格式如下：

```
var myfunction = function name(parameter_list){
  // 函数中的代码
};
```

参数说明如下：

- myfunction：变量名，可以通过它来调用等号之后的函数；
- name：函数名，可以省略（一般情况下我们也会将其省略），如果省略那么该函数就会成为一个匿名函数；
- parameter_list：为参数列表，一个函数最多可以有 255 个参数。

示例代码如下：

```
// 函数声明
function getSum(num1, num2) {
var total = num1 + num2;
return total;
}
// 函数表达式
var getSum = function(num1, num2) {
var total = num1 + num2;
return total;
};
```

上面示例中的两个函数是等价的，它们的功能、返回值、调用方法都是相同的。

注意：在函数声明中，不需要在右花括号后放置分号，但若使用函数表达式就应该在表达式的最后以分号结尾。

函数声明和函数表达式虽然看起来非常相似，但它们的运行方式是不同的，如下例所示：

```
declaration();          // 输出: function declaration
function declaration() {
document.write("function declaration");
}
expression();           // 报错: Uncaught TypeError: undefined is not a function
var expression = function() {
document.write("function expression");
};
```

如上例所示，如果函数表达式在定义之前被调用，会抛出异常（报错），但函数声明则可以成功运行。这是因为在程序执行前，JavaScript 会先对函数声明进行解析，因此无论是在函数声明前还是声明后调用函数都是可行的。而函数表达式则是将一个匿名函数赋值给一个变量，所以在程序还没有执行到该表达式之前，相当于函数还未定义，因此无法调用。

js 输入输出语句

某些情况下，我们可能需要将程序的运行结果输出到浏览器中，JavaScript 中为我们提供了多种不同的输出语句来向浏览器中输出内容：

1. 使用 alert() 函数来弹出提示框；
2. 使用 confirm() 函数来弹出一个对话框；
3. 使用 prompt() 浏览器弹出输入框，用户可以输入；
4. 使用 console.log() 在浏览器的控制台输出内容。
5. 使用 document.write() 方法将内容写入到 HTML 文档中；
6. 使用 innerHTML 将内容写入到 HTML 标签中；

1. alert() 函数

使用 JS alert() 函数可以在浏览器中弹出一个提示框，在提示框中我们可以定义要输出的内容，语法格式如下：

alert(message);

其中 message 为要在提示框中输出的内容，需要注意的是，alert() 中只能输出文本内容。

alert() 函数是 window 对象下的一个函数，所以有时为了代码更严谨，我们也可以使用 window.alert() 的形式来调用 alert() 函数。

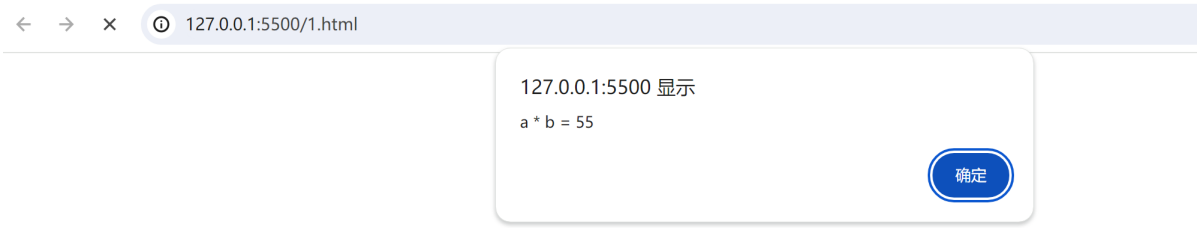
示例代码如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>JavaScript</title>
</head>

<body>
    <script type="text/javascript">
        var a = 11, b = 5;
        window.alert("a * b = " + a * b);
    </script>
</body>

</html>
```

运行结果如下图所示：



2. confirm() 函数

JS confirm() 函数与 alert() 函数相似，它们都是 window 对象下的函数，同样可以在浏览器窗口弹出一个提示框，不同的是，使用 confirm() 函数创建的提示框中，除了包含一个“确定”按钮外，还有一个“取消”按钮。如果点击“确定”按钮，那么 confirm() 函数会返回一个布尔值 true，如果点击“取消”按钮，那么 confirm() 函数会返回一个布尔值 false。

示例代码如下：

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>JavaScript</title>
</head>

<body>
    <script type="text/javascript">
    var res = window.confirm("这里是要显示的内容");
    if (res == true) {
        alert("你点击了“确定”按钮");
    } else {
        alert("你点击了“取消”按钮");
```

```
    }
    </script>
</body>

</html>
```

运行结果如下图所示：

3. prompt()

prompt()是 JavaScript 中的一个内置函数，用于显示一个对话框，这个对话框包含一个文本消息（由 prompt()函数的第一个参数指定）以及一个文本输入字段，用户可以在这个字段中输入数据。prompt()函数会返回用户输入的数据，如果用户取消对话框，则返回 null。

示例代码如下

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>JavaScript</title>
</head>

<body>
    <script type="text/javascript">
    var userInput = prompt("请输入你的名字：", "例如:张三");

    if (userInput !== null) {
        alert("你好，" + userInput + "! 欢迎来到这个网站。");
    } else {
        alert("你没有输入名字，或者取消了操作。");
    }
    </script>
</body>

</html>
```

运行代码如下图所示

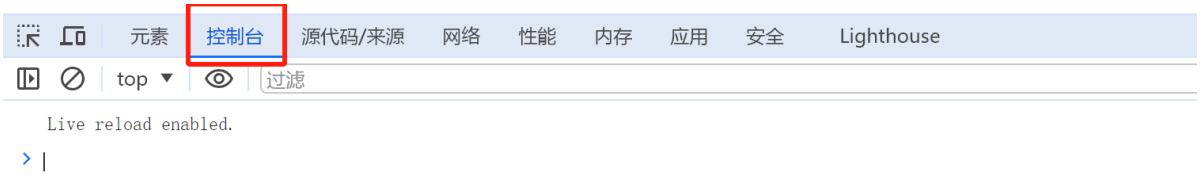
4. console.log()

使用 JS console.log() 可以在浏览器的控制台输出信息，我们通常使用 console.log() 来调试程序，其语法格式如下：

console.log(message);

其中 message 为要输出的内容，可以是字符串或者对象类型。与 window.alert() 和 window.confirm() 可以分别简写成 alert() 和 confirm() 不同，console.log() 不能简写。

要看到 console.log() 的输出内容需要先打开浏览器的控制台。以 Chrome 浏览器为例，要打开控制台您只需要在浏览器窗口按 F12 快捷键，或者点击鼠标右键，并在弹出的菜单中选择“检查”选项即可。最后，在打开的控制台中选择“控制台”选项，如下图所示：



示例代码如下：

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>JavaScript</title>
</head>

<body>
    <script type="text/javascript">
        var myArr = ["Chrome", "Firefox", "Edge", "Safari", "Opera"];
        console.log(myArr);
    </script>
</body>

</html>
```

运行结果如下图所示：

5. document.write()

使用 JS document.write() 可以向 HTML 文档中写入 HTML 或者 JavaScript 代码，语法格式如下：

document.write(exp1, exp2, exp3, ...);

其中 exp1、exp2、exp3 为要向文档中写入的内容，document.write() 可以接收多个参数，即我们可以一次向文档中写入多个内容，内容之间使用逗号进行分隔。

示例代码如下：

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>JavaScript</title>
</head>

<body>
    <script type="text/javascript">
        document.write("<p>现在的时间是：</p>");
        document.write(Date());
    </script>
</body>

</html>
```

运行结果如下图所示：

6. innerHTML

与前面介绍的几个函数不同，innerHTML 是一个属性而不是一个函数，通过它可以设置或者获取指定 HTML 标签中的内容，示例代码如下：

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>JavaScript</title>
</head>

<body>
    <div id="demo">JavaScript 输出</div>
    <script type="text/javascript">
        var demo = document.getElementById("demo");
        console.log(demo.innerHTML);
        demo.innerHTML = "<h2>innerHTML</h2>"
    </script>
    <!-- <script type="text/javascript">
        var demo = document.getElementById("demo");
        console.log(demo.textContent);
        demo.textContent = "<h2>innerHTML</h2>"
    </script> -->
</body>

</html>
```

运行结果如下图所示：

JavaScript 脚本语言

Javascript，JScript，ActionScript等脚本语言都是基于ECMAScript标准实现的。在JavaScript，JScript和ActionScript中声明变量，操作数组等语法完全一样，因为它们都是ECMAScript。但是在操作浏览器对象等方面又有各自独特的方法，这些都是各自语言的扩展。

JavaScript是由ECMAScript，DOM和BOM三者组成的。

JavaScript语言简称JS。

BOM

BOM(Browser Object Model)即浏览器对象模型，它提供了独立于内容而与浏览器窗口进行交互的对象，其核心对象是window。

BOM中共有五大对象

- **Window**：窗口对象

```
window.open() - 打开新窗口
window.close() - 关闭当前窗口
window.alert() - 弹框
```

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>JavaScript</title>
</head>

<body>
    <script type="text/javascript">
        // window.open("https://www.baidu.com"); //打开新窗口
        // window.close(); //关闭当前窗口
```

```
        // window.alert(1); //弹框
    </script>
</body>

</html>
```

- **Navigator**：浏览器对象

```
navigator.appCodeName  - 浏览器代号
navigator.appName  - 浏览器名称
navigator.appVersion - 浏览器版本
navigator.cookieEnabled - 启用Cookies
navigator.platform - 硬件平台
navigator.userAgent - 用户代理
navigator.language - 用户代理语言
```

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>JavaScript</title>
</head>

<body>
    <script type="text/javascript">
        document.write("navigator.appCodeName:" + navigator.appCodeName + "
<br>");
        document.write("navigator.appName:" + navigator.appName + "<br>");
        document.write("navigator.appVersion:" + navigator.appVersion + "<br>");
        document.write("navigator.cookieEnabled:" + navigator.cookieEnabled + "
<br>");
        document.write("navigator.onLine:" + navigator.onLine + "<br>");
        document.write("navigator.platform:" + navigator.platform + "<br>");
        document.write("navigator.userAgent:" + navigator.userAgent + "<br>");
        document.write("navigator.javaEnabled():" + navigator.javaEnabled() + "
<br>");
    </script>
</body>

</html>
```

- **Screen**：显示器屏幕对象

```
screen.availWidth - 可用的屏幕宽度
screen.availHeight - 可用的屏幕高度
```

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>JavaScript</title>
```

```
</head>

<body>
    <button onclick="myBack()">back()</button>
    <button onclick="myForward()">forward()</button>
    <script type="text/javascript">
        document.write(screen.availHeight + "<br>");// 输出：1050
        document.write(screen.availWidth + "<br>"); // 输出：1920
    </script>
</body>

</html>
```

- **Histroy**：历史记录对象

```
history.back() - 与在浏览器点击后退按钮相同
history.forward() - 与在浏览器中点击向前按钮相同
```

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>JavaScript</title>
</head>

<body>
    <script type="text/javascript">
        document.write(screen.availHeight + "<br>");// 输出：1050
        document.write(screen.availWidth + "<br>"); // 输出：1920
    </script>
</body>

</html>
```

- **Location**：地址栏对象

```
location.hostname - 返回 web 主机的域名
location.pathname - 返回当前页面的路径和文件名
location.port  - 返回 web 主机的端口 （80 或 443）
location.protocol - 返回所使用的 web 协议（http: 或 https:）
location.href - 返回当前页面的 URL
location.assign() - 传递一个url参数，打开新url，替换当前页面，并在浏览记录中生成一条记录，页面替换后可以返回上一个页面。
location.replace() - 参数为一个url，打开新url，替换当前页面，不会在历史记录中生成新记录，替换页面后无法再次返回之前页面。
location.reload() - 重新加载当前显示的页面，参数可以为boolean类型，默认为false，表示以最有效方式重新加载，可能从缓存中直接加载。如果参数为true，强制从服务器中重新加载。
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <title>JavaScript</title>
</head>
<body>
    <script type="text/javascript">
        document.write("<b>hostname:</b>" + location.hostname + "<br>");
        document.write("<b>pathname:</b>" + location.pathname + "<br>");
        document.write("<b>port:</b>" + location.port + "<br>");
        document.write("<b>port</b>" + location.protocol + "<br>");
        document.write("<b>protocol:</b>" + location.href + "<br>");
        //location.assign("http://www.baidu.com")
        //location.replace("http://www.baidu.com")
        //location.reload();
    </script>
</body>
</html>
```

DOM

我们前面学习的都是JS语言核心部分，也就是ECMAScript。一般都是在控制台、输出语句里操作，JS还包括DOM和BOM。

DOM（Document Object Model，文档对象模型）描绘了一个层次化的节点树，允许开发人员添加、移除和修改页面的某一部分。这使得JavaScript操作HTML，不是在操作字符串，而是在操作节点，极大地降低了编程难度。

DOM对很多东西做了抽象，提供了丰富的API：取得元素、css样式、事件、运动、元素尺寸位置、节点操作等等。

HTML 操作

document：表示文档（表示整个页面）对象。document对象具有页面几乎所有的方法或者属性。

读取： document.title 页面的标题。

赋值：使用=进行赋值。

一般操作元素都是从获取元素开始的。

获取元素的方法： getElementById()　　通过id属性获取元素对象。

通过id属性得到的元素对象的数据类型是对象

通过元素对象操作属性

读取：可以通过对象的点方法得到属性名。

设置：用=进行赋值

点语法只能读取或者设置元素的自带有属性不能设置读取元素的自定义属性。

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>JavaScript</title>
</head>
```

```
<body>
    <div id="demo">JavaScript 输出</div>
    <script type="text/javascript">
        var demo = document.getElementById("demo");
        console.log(demo.innerHTML);
        demo.innerHTML = "<h2>innerHTML</h2>"
    </script>
    <!-- <script type="text/javascript">
        var demo = document.getElementById("demo");
        console.log(demo.textContent);
        demo.textContent = "<h2>innerHTML</h2>"
    </script> -->
</body>

</html>
```

常见dom属性

```
document.cookie      设置或返回与当前文档有关的所有 cookie
document.body    返回文档的body元素
document.domain      返回当前文档的域名
document.title   返回当前文档的标题
document.URL     返回文档完整的URL
document.write()     向文档写 HTML 表达式 或 JavaScript 代码
```

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>JavaScript</title>
</head>

<body>
    <script type="text/javascript">
        document.write(document.cookie + "<br>");
        document.write(document.body + "<br>");
        document.write(document.domain + "<br>");
        document.write(document.title + "<br>");
        document.write(document.URL + "<br>");
    </script>
</body>

</html>
```

DOM 事件

事件监听：我们计算机在解析我们JS代码的时候，会去看某一些元素身上是否添加了事件。并监听这些事件有没有被触发，如果触发就立即执行相应的行为。

> onclick 单击
>
> ondblclick 双击
>
> onmouseenter 鼠标进入

onmouseleave	鼠标离开
onmousedown	鼠标按下
onmouseup	鼠标弹起
onfocus	获取焦点
onblur	失去焦点
onload	加载完毕之后

元素绑定事件：元素.事件名 = fn。可以绑定匿名函数或者函数名（千万不要在函数名后面书写小括号）。

注意：

我们在body中书写js时，需要将js书写在所有html元素之后。当html元素加载完毕之后在执行js。

如果js书写在head标签中，必须书写onload事件，window.onload表示当html元素加载完毕之后执行内部的语句。

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>JavaScript Event Handlers</title>
    <script>
        function handleClick() {
            alert('Clicked!');
        }
        function handleDoubleClick() {
            alert('Double Clicked!');
        }
        function handleMouseEnter() {
            alert('Mouse entered the element!');
        }
        function handleMouseLeave() {
            alert('Mouse left the element!');
        }
        function handleMouseDown() {
            alert('Mouse button pressed down!');
        }
        function handleMouseUp() {
            alert('Mouse button released!');
        }
        function handleFocus() {
            alert('Element got focus!');
        }
        function handleBlur() {
            alert('Element lost focus!');
        }
        function handleImageLoad() {
            alert('Image loaded successfully!');
        }
    </script>
```

```
</head>

<body>
    <!-- Input tag with event handlers -->
    <input type="text" id="myInput" onclick="handleClick()"/>
    <input type="text" id="myInput" ondblclick="handleDoubleClick()"/>
    <input type="text" id="myInput" onmouseenter="handleMouseEnter()"/>
    <input type="text" id="myInput" onmousedown="handleMouseDown()"/>
    <input type="text" id="myInput" onmouseup="handleMouseUp()"/>
    <input type="text" id="myInput" onfocus="handleFocus()" />
    <input type="text" id="myInput" onblur="handleBlur()" />
    <!-- Image tag with onload event handler -->
    <img id="myImage" src="1.png" onload="handleImageLoad()" alt="An example
image" />

</body>

</html>
```