目录

- 常量与变量
  - **常量** : 值不可以改变的叫常量。 （`const`）
    - 变量的*命名*: 数字/字母/下划线/$;不能以数字开头; 驼峰式或下划线写法
  - **变量** : 类似数学中的x,y; 可以改变的量 (`let`, `var`)
- 进制转换
  - 十进制转换为其他进制: **模2取余**、模8取余、模16取余
  - 其他进制转为十进制: 从右到左,**每个数**依次 **× 进制数 × 从0开始的幂数**
    - 二转十： 1100 => 0×2^0 + 0×2^1 + 1×2^2 + 1*2^3
- 运算符
  - 算数运算符: **+ - * / %**
  - 关系运算符: **> < >= <= != === !==**
  - 逻辑运算符: **&& || !**
  - 一元运算符: **++ --**
  - 赋值运算符: *= += -= = /= %=
- 强制数据类型转换: `Boolean()` `Number()` `String()`
  - `parseInt()`: 字符串取整, 开头必须是数字， 提取到不是数字为止, 开头不是数字为`NaN`
  - `pareseFloat()`: 和 `parseInt` 区别, 会提取小数点
- **表达式**: **任何数据与运算符组成的式子**
- 前言: JS的数据类型: `[Number,String,Boolean,NaN,Undefined,Object,Symbol]`

1. 什么是 javascript?

- 官方概念：这是一个 *跨平台* 的 *脚本* 语言。
  - 平台：就是运行环境，这里一般指操作系统。
    - 跨平台：在各种环境下都可以运行。
  - 脚本语言：脚本语言的特点，不能独立运行，要依赖于网页。
    - HTML的运行离不开浏览器。
    - JS程序的运行离不开HTML页面。

  可以看互联网巨头页面发展:http://web.archive.org/ (需墙)

2. javascript的组成

1. ECMAScript 3 , 4 , 5 , 6 , 7
2. DOM 文档
3. BOM (一个浏览器窗口代表一个BOM)

- 注 : 所有的JS代码在一个script标签中编写
- 属性:
  1. `type = 'text/javascript'` 声明当前标签的文本格式。 （可以省略）
  2. `src = 'demo.js'` 引入外部的.JS文件
     - 注：link引入css是用的是href，引入JS用的是src。

- 注意点：
  1. 可以有多个`script`标签，多个`script`标签是**自上而下顺序执行**的。
  2. 一个`script`标签只能专心的做一件事。(引入文件或在标签内写代码)

3. 向页面输出内容：

1. `alert('这是一些内容');` 在页面上弹出警告框
2. `document.write('这是一些内容');` 当前页面上输出内容
   - 标签的解析 (可以解析标签)
   - 转义字符 （& and符号）
     - `<` 代表 `<`
     - `>` 代表 `>`
3. `console.log('这是一些内容');`
   - 在浏览器调试面板控制台输出内容。
   - 一般用在代码调试
     - 小彩蛋：招聘简介放在这里。
   - `console.dir(对象)` 可以显示一个对象的所有属性和方法

- 注：在编写JS代码的时候，要在每一行代码结束的位置加 `;` (分号)
- 代码压缩：去掉编写代码的时候，所有的空格、tab键和换行。

4. JS注释

```
// 单行注释  -> ctrl + /  快捷键
```

```
/*
  多行注释
  ctrl + shift + /  快捷键（华为matebook14快捷键为alt + shift + a）
*/
```

5. 常量

- **常量** : **值不可以改变**的叫常量。
- 数据类型 :
  - **基本数据类型** :
    1. 数字
       - 类型 : `Number`
       - 例子 : 100 -20 3.14
    2. 布尔值
       - 类型 : `Boolean`
       - 例子 : true(真) false(假)
    3. 字符串

        - 类型：`String` 所有带单引号或者双引号的都叫字符串。（单引号/双引号效果一样，必须成对出现）
        - 例子：'hello' "world" '100' "3.14"
    - **复合 / 引用数据类型**：`Object`
        - 函数、数组、对象...都属于`Object`
    - **特殊数据类型**：
        - `null` 空
        - `undefined` (代表一种状态)
    - `NaN` (not a number)不是一个数字
        - NaN代表一种状态,不是数据类型

6. 变量

- **变量**：**值可以被修改**的叫做变量。类似于我们数学中的X和Y。
- 计算机的组成 :磁盘、内存、CPU/GPU (主要是用来运算的)
    - 【注】程序执行：**程序从磁盘被读取到内存中,被CPU运行。**
        - 内存是有空间的，程序本质上运行是在内存中运行的。
        - 【注】编程就是合理的分配内存。

1. 声明变量(必须声明以后才能使用)：`var name = 'zs'`
2. 初始化：声明变量时，直接给这个变量赋值。
3. 声明变量时，没有值赋给这个变量，系统默认赋值成`undefined`。

```
// 1.初始化，声明变量时赋值
    var num = 10;  // 在内存中划出一块空间(num),并且赋值常量10。
    alert(num);    // 最后弹出num。

// 2.声明变量时不赋值 (不推荐，如果没有值，要赋值为null)
    var num;         // 划出一块空间,没有直接赋值,被系统赋值为undefined
    num = 10;        // 然后把undefined擦除,重新赋值为10
    alert(num);      // 最后弹出num。

// 3.声明变量时赋值null
    var num null;    // 在内存中划出一块空间,赋值为null
    num = 10;        // 再次赋值时,是直接写入10，没有擦除操作。
    alert(num);
```

【注】将声明变量 在没有值赋给这个变量的时候，**默认赋值为NULL**，提高程序的执行效率。

7. 变量的命名。

- 标识符：所有用户自定义的名字叫做标识符。
    - 变量名也是标识符。
- 标识符 **命名规则**：
    - 只能由数字、字母、下划线、美元符号($)组成。
    - 不能以数字开头。

    - 不能是保留字和关键字。
        - 关键字：系统征用的有特殊功能的单词。
        - 保留字：js已经征用，以后可能有用的单词叫做保留字。
    - 大小写敏感 age Age 这是两个完全不同的变量。
    - 见名思意。(尽量使用英文全称)
    - 单词个数超过两个的
        - 驼峰式： `className`
        - 下划线命名: `class_name`
- **变量** : **弱引用类型**。赋值成什么数据类型就是什么数据类型。
- `typeof` 关键字
    - 格式: `typeof 常量/变量`
    - 功能: 输出当前常量或者变量的数据类型。

```
alert(typeof 100);       // 弹出对话框为 number
alert(typeof 'hello')    // 弹出对话框为 string
alert(typeof undefined)  // 弹出对话框为 undefined
alert(typeof true/false) // 弹出对话框为 boolean

alert(typeof typeof 100) // 弹出对话框为 string
alert(typeof typeof undefined) // 弹出对话框为 string
```

        - 在参数为数组，对象或者null时，typeof返回的结果都是object

8. 进制转换。

- 十进制： 0~9
  二进制： 0~1
  八进制： 1~8
  十六进制: 1~E

- **十**进制 **转 二**进制: **模二取余**,固定流程
    - 数值÷2，余数留下，没有余数则留0，一直除到商为0，然后自下而上把余数连起来。
        - 十进制 52 转二进制 是110100
- 二进制 110100 转十进制

```
0*2^0 + 0*2^1 + 1*2^2 + 0*2^3 + 1*2^4 + 1*2^5 = 0+0+4+0+16+32=52
```

- **十**进制 **转 八**进制 : **模八取余**
- 八进制 2764 转十进制

```
4*8^0 + 6*8^1 + 7*8^2 + 2*8^3 = 4+48+448+1024=1524
```

- **十**进制 **转** 十**六**进制 : **模十六取余**

- 十六进制转十进制同理

- **二**进制 **转 十六**进制 => `2^4 = 16` 所以是四位一组
  规则 : 从右往左数，每四位一组，不足三位的用0补齐，将每组数单独转成十六进制。
- 二进制转8进制: 三位一组，将每组数单独转成八进制

9. 运算符。

1. 算数运算符
   `+ - * / %`
   - `%` (取余符号) `5%` 不能表示百分之5,可以用 `0.05` 来表示 `5%`。

     ```
     var num = 5;
     alert(num % 3);   // 2
     ```

2. 关系运算符
   `> < >= <= ==(等于) !=(不等于) ===(恒等) !==(恒不等)`
3. 逻辑运算符
   `&&(与) ||(或) !(非)`
4. 一元运算符
   `++ --`
5. 赋值运算符
   基本赋值运算符 `=`
   复合赋值运算符 `+= -= *= /= %=`

- **表达式** : **任何数据和运算符组成的式子**叫做表达式。
  - 注意 表达式的值 和 表达式的功能

- 算数运算符
  - 【注】将不同数据类型进行算数计算，会进行数据类型转换
    - **自动数据类型转换** : 不同数据类型之间是没有办法进行运算,将数据转换为同一类型,再进行计算。
      1. **字符串和任何数据做 `+` 运算**，会进行**字符串拼接**。

         ```
         alert("你" + "好");     //  '你好'
         alert("1" + 2);         //   '12'
         alert("1" + null)       //   '1null '
         alert("1" + true)       //   '1true'
         alert("1" + undefined) //   '1undefined'
         alert("1" + NaN)        //   '1NaN'    上述都为字符串
         ```

2. **字符串和任何数据做 `+` 以外运算,字符串要先转成数字，再进行运算。**
   - 纯数字字符串，会转换成对应的数字。不是纯数字字符串，会转成 `NaN`。 `NaN` 和字符串 相加 会进行字符串拼接，其他运算都是 `NaN`。
     - 【注】 `NaN` 是一个值，类型是 `number`。意思是不是一个数字。

```
alert(2-'2');          // 0    类型为number
alert(2-'wo');         // NaN  类型为number
alert(true - 'wo');    // NaN  类型为number
alert("2" - true);     // 1    类型为number
alert(undefined-'wo'); // NaN  类型为number
```

3. **除字符串以外的数据，在进行算数运算的时候，先转成数字，再进行运算。**

```
true => 1   false => 0   null => 0   undefined => NaN
```

```
alert(10 + true);    //结果   11   类型为number
alert(10 - false);   //结果   10
```

10. 算数运算符的细节。

- 计算机是不会进行小数运算。(天生有Bug)
  - 例子：alert(0.8 - 0.1); //结果 0.7000000000000001
  - 小彩蛋：金融账户，只存储整数，单位是分。
- 在JS中除数可以为0

```
alert(10 / 0);   //结果 infinity  无穷大
alert(-10 / 0);  //结果 -infinity 无穷小
alert(0 / 0);    //结果 NaN
```

  - 【注】infinity也遵循算术运算规则，类型为number

```
alert(Infinity + 'hello'); //结果 拼接字符串
alert(Infinity - 100);  //结果  infinity(无穷大)
```

11. 赋值运算符 与 一元运算符。

- 普通赋值运算符 : `=`
  - **作用** : 将 `=` **右边的值赋给左边的变量。**

```
var num = 10 + 20;  //先计算右边的值，为30，再给左边变量赋值。
num = num + 10;     //先计算右边的值，为40，再给左边变量赋值。
alert(num);         //num: 40
```

  - 复合赋值运算符： `+= -= *= /= %=`

```
num += 5;  /* 等于 */  num = num + 5;
```

- 一元运算符：`++ --`
  - `++` 功能：对原有的变量进行+1操作。
  - 写法：`a++` 、 `++a`

```
var a = 5;
a++;
alert(a);   //a: 6
```

  - `a++`：++**后置**，**先取a**的值**作为a++表达式的值**，然后**再对a进行+1**。
    - a++ 是先取值，再执行++运算。
  - `++a`：++**前置**，**先对a** 进行 **+1** 操作，**再取a** 的值作**为++a表达式的值**。
    - ++a 是先执行++运算，再取值。

```
var a = 10;
alert(a++ + ++a + a + ++a + a++ + a); //结果 10+12+12+13+13+14=74
```

- `--` 功能：对原有的变量进行-1操作。与 `++` 相反
- 写法：`a--` 、 `--a`

12. 强制数据类型转换。

- `Boolean()` ： 将其他数据类型强制转换成布尔值

```
//数字转布尔值（非0即真）
alert(Boolean(0));          //false
alert(Boolean(3.14));       //true

//字符串转布尔值（非空即真）
alert(Boolean("11"));       //true
alert(Boolean("hello"));    //true
alert(Boolean(""));         //空字符串,false

//特殊数据类型转布尔值
alert(Boolean(null));       //false
alert(Boolean(undefined));  //false
alert(Boolean(NaN));        //false
```

- `Number()` ： 将其他数据类型强制转换为数字

```
//字符串转数字（只有纯数字组成的字符串，才能转换为数字，否则为NaN）
alert(Number("100"));       // 100
alert(Number("100a"));      // NaN

//布尔值转数字（true为1，false为0）
Number(true) == 1;  Number(false) == 0;

//特殊数据类型转数字
alert(Number(null));        // 0
alert(Number(undefined));   // NaN
alert(Number(NaN));         // NaN
```

- `String()` ：将其他数据类型转换为字符串
- `parseInt()` ：对数字或字符串取整

```
//字符串取整 (开头有数字则提取，提取到不是数字的位置为止，提取不了则为NaN)
parseInt("100.5a");  // 100
parseInt("10b0a");   // 10
parseInt("b100a");   // NaN
parseInt("hello");   // NaN
```

  - 特殊功能，`parseInt()` **其他进制转十进制**，必须**传入字符串**

```
52 => 二进制:110100;八进制:64;十六进制:34;
//二进制转十进制
var str1 = "110100";
alert(parseInt(str1,2));

//八进制转十进制 => 十六进制和上述同理
var str2 = "64"
alert(parseInt(str2,8))
```

- `parseFloat()` ：对数字或字符串取浮点数

```
//字符串取浮点数 （和parseInt的区别:会取小数位）
parseFloat("100.5a");  // 100.5
parseFloat("10b0.5a"); // 10
parseFloat("b100a");   // NaN
parseFloat("hello");   // NaN
```

13. 关系运算符

- `> < >= <= ==(等于) !=(不等于) ===(恒等) !==(恒不等)`
  - `=` 是赋值，`==`是比较
  - 关系运算符最后运算的值，绝对是布尔值。(成立为true，不成立为false)
- **自动数据类型转换**：当关系运算符操作**非数值时**
  - 两个操作数**都是字符串**，则**逐一比较**两个字符串对应的**字符编码值。**
    - 字符编码:**ASCII**码，是电脑内部每一个字符和字符对应编码的一张表。
    - 如果是两个单个字符进行比较，直接比较字符的ASCII码值。
    - **两个或以上的字符串**进行比较，会**逐一比较**，如果**比较出大小直接得出结果。**

```
alert("abcd" > "adc");   //结果为false
// 先进行第一个字符比较 : a 和 a    //结果相等，进行下一个字符串
// 第二个字符进行比较 : b 和 d     //结果b的ASCII码小于d
// 得出 : 第一个字符串是 小于 第二个字符串的，结果为假。
```

  - 两个操作数中**有一个数值**，则**将另一个转换为数值**，再**进行比较。**
    - 【注】`NaN` **和任何数比较都不相等**(包括NaN)
  - **除数值以外的数**，两个**类型不一样**进行比较时，**先转成数字**再进行比较。
    - 【注】两个类型一样，直接看值一不一样。

- 在相等(`==`)和不等(`!=`)的比较上，如果操作数是**非数值**，则遵循以下规则
  1. **一个操作数是布尔值**，比较前将其**转换为数值。**
  2. **一个操作数是字符串**，比较前将其**转换为数值**再比较。
  3. 一个操作数是`NaN`,则`==`返回`false`，`!=`返回`true`;并且`NaN`**和自身不等。**
  4. `===`**恒等**，必须**数值和数据类型都相等才能返回true**，否则返回false。

```
alert('10' === 10);         // false
alert(Number('10') === 10); // true
```

- 特殊值：

```
null == undefined //true   重点
'NaN'  ==  NaN    //false
5      ==  NaN    //false
NaN    ==  NaN    //false  重点
NaN    !=  NaN    //true
false  ==  0      //true
true   ==  1      //true
undefined == 0    //false
null   ==  0      //false     虽然null转数字为0，但在比较中不等
'100' ==  100     //true
'100' === 100     //false

Number(null);         //0
Number(undefined);    //NaN
null == undefine;   //true
```

14. 逻辑运算符

- `&&(与) ||(或) !(非)`
- `&&(与)`：表达式1 `&&` 表达式2
  - **两个表达式都为true**时，整个表达式**才为true**

```
alert(10 > 5 && 4 < 6);   // true
```

  - 短路操作：当表达式1为false时，表达式2就不执行了，直接得出结果false。

```
alert(num)  //没有声明变量直接输出会在控制台报错
alert(10<5 && alert(num)); //弹出false，控制台并没有报错。
alert(10>5 && alert(num)); //不弹出，控制台报错。
```

- `||(或)`：表达式1 `||` 表达式2
  - 当其中**一个表达式为true**时，整个表达式**就为true**，两个都为false时，整个表达式才能为false

```
alert(10>5 || 5>10);    // true
alert(10<5 || 5>10);    // false
```

  - 短路操作：当表达式1为true时，表达式2就不执行了，直接得出结果true。

- `!(非)`： `!表达式1`
  - **将表达式的值转换成布尔值，再进行取反。**
    - number非0即真，字符串非空即真, NaN为false

```
alert(! 10 > 5);   //相当于 !true 结果为 false
```