02 第一个程序Hello World

> 学了三个指令

```
//弹出警告的窗口
alert("warning!!!");
//在文档写入内容，会在窗口显示
document.write("hello world");
//在控制台写入内容
console.log("wowo");
```

03 JS编写的位置

> JS引用样式的三种方式，和CSS很像

- 写在body里，例如：
  - 写在button标签的onclick属性中，当触发的时候就会执行
  - 写在a标签的href属性中，点击超链接就会执行
- 内部样式
  - 写在head里内部引用
- 外部样式
  - script标签引入

```
<script type="text/javascript" src="">
//这里即使再写也没用
</script>
```

04 JS基本语法

- 注释

```
//单行注释

/*
多行注释
多行注释
*/
```

- JS中严格区分大小写
- JS每条语句以分号结尾
- JS中会忽略多个空格和换行

05 字面量和变量

字面量：例如1 2 3 4，都是一些不可以改变的值，可以直接使用的，但是我们一般不会直接使用字面量，使用起来不方便

变量：变量可以保存字面量，而且变量可以任意改变，使用方便，开发的时候基本都是用变量去存储字面量，很少直接使用字面量，通过变量也可以对字面量进行描述

- 变量的使用步骤，var只用在第一次声明变量的时候写

- 声明变量：var 变量名；
- 初始化变量：变量 = ；
- 或者声明和初始化同时进行：var 变量名 = ；

06 标识符

- 变量名、函数名、属性名都属于标识符
- 标识符的命名规则：
  - 标识符可以包含字母、数字、_、$
  - 标识符不能以数字开头
  - 标识符里不能是ES中的关键字或者保留字
  - 建议使用驼峰命名法（首字母小写，每个单词的开头字母大写其余小写）
- 中文也可以作为标识符，但是不建议用

07 字符串

数据类型（字面量的类型）

JS中一共有6中数据类型前5种是基本数据类型，最后一种是引用数据类型

```
String  字符串
Number  数值
Boolean 布尔值
Null    空值
Undefined   未定义
Object  对象
```

- String 字符串

```
var str = "hello";  //字符串必须放在引号里（单引号或者双引号都行）
```

注意引号不能嵌套

- 转义字符

```
\"  双引号
\n  换行
\t  制表符
\\  \
```

08 Number

> typeof运算符的使用；Number . MAX_VALUE Number . MIN_VALUE

注意JS是顺序执行

- 在JS中所有的数值都是Number类型

```
var a = 123;
var b = "123";
```

- 怎样区分上面这两个变量的类型？运算符typeof

```
var a = 123;
typeof a;
var b = "123"
typeof b;
```

- JS中表示数值的最大值和最小值（输出都是科学计数法）

```
Number.MAX_VALUE;
Number.MIN_VALUE;
```

- 使用typeof检查NaN的数据类型也是Number

```
var a = "andn";
var b = "bas";
var c = a * b;
console.log(c); //输出 Not a Number
```

在JS里整数的运算基本都没有问题，如果计算的是浮点数，可能会造成精度的问题

09 布尔值

布尔值只有两个true和false，主要是进行逻辑判断

```
var bool1 = true;
var bool2 = false;
console.log(bool1);
console.log(bool2);
```

10 Null 和Undefined

- Null数据类型只有一个值null
- null的作用是去表示一个空的对象
- 使用typeof检查null的数据类型时会返回object

```
var a = null;
console.log(typeof a);
```

- Undefined数据类型的值只有一个就是undefined
- 当声明变量但是没有初始化的时候这个变量的数据类型就是Undefined

```
var a;
console.log(a);
console.log(typeof a);
//都会返回undefined
```

11 强制类型转换

将一个数据类型强制转化为另外一种数据类型

一般是将其他数据类型转换为String、Number、Boolean

其他数据类型转换为String数据类型

- 方法一：调用被转换数据类型的toString( )方法

> 这种方法不会影响原来的变量，它会将转换的结果返回
>
> 注意null和undefined没有toString的功能

```
var a = 123;
a = a.toString();
console.log(typeof a);
console.log(a);
```

```
var b = true;
b = b.toString();
console.log(typeof b);
console.log(b);
```

- 方法二：使用String( )函数直接将待转换的变量作为参数传入String( )函数中

> 对于number和Boolean实际上是调用toString( )方法
>
> 但是对于null和undefined，就不会调用toString( )方法
>
> null ->"null"
>
> undefined ->"undefined"

```
var a = 123;
var a = String(a);
console.log(typeof a);
console.log(a);
```

```
var b = true;
b = String(b);
console.log(typeof b);
console.log(b);
```

其他数据类型转换为Number数据类型

- 方法一：使用Number( )函数

> 字符串里只要存在非数值类型就转不了Number，例如"123px"，返回的是NaN，所以这是这种方法的缺陷
>
> 如果被转换的字符串是一个空串或者全是空格的字符串，则返回0
>
> true->1
>
> false ->0
>
> null ->0
>
> undefined ->NaN

```
var a = "123";
a = Number(a);
console.log(typeof a);
console.log(a);
```

```
var = "adc12";
a = Number(a);
console.log(typeof a);
console.log(a);
//返回 NaN
```

- 方法二：parseInt( ) parseFloat( ) 这种方法专门用来对付字符串，非字符串返回NaN
- parseInt( )将字符串里从左往右的有效整数提取出来，遇到非数字就停止，例如"a15443"，返回的是NaN；也可以用该函数进行取整操作

```
var a = "123px";
a = parseInt(a);
console.log(typeof a);
console.log(a);
```

- parseFloat( )将字符串里从左往右的有效小数提取出来，遇到非数字就停止

```
var a = "123.59px";
a = a.parseFloat(a);
console.log(typeof a);
console.log(a);
```

其他类型转换为Boolean

使用Boolean( )函数

> 数字转布尔，除了0和NaN都是true
>
> 字符串转布尔，除了空串，其余都是true
>
> null ->false
>
> undefined ->false
>
> object ->true

```
var b = 123;
b = Boolean(b);
console.log(typeof b);
console.log(b);
```

13 其他进制的数字

浏览器输出都是10进制

```
//  0x开头的是16进制
//  0开头的是8进制
//  0b开头是2进制
```

像"070"这种字符串，在转换为数字后，有些浏览器会当成8进制解析，有些浏览器会当成10进制解析，为了统一解析的规则，可以使用parseInt( )函数的第二个参数规定转换为多少进制

```
var a = "070";
a = parseInt(a,10); //10表示转换为10进制
console.log(typeof a);
console.log(a);
```

15 运算符

> 加法自动转换为字符串类型
>
> 减法、乘法、除法自动转换为数值类型
>
> 隐式类型转换，加法的字符串类型转换
>
> 减法-0，乘法*1，除法/1的数值类型转换

typeof就是一个运算符，作用是获得变量的数据类型，返回的数据类型是一个String类型

算数运算符

```
+
-
*
/
%
```

加法

任何值和NaN加法运算，结果都是NaN

多个字符串的运算：拼接操作

字符串和值的运算：数值先转换为字符串再做拼接运算

利用字符串和数值的运算把数值转换为字符串（隐式类型转换，由浏览器自动完成，实际上也是调用了String( )函数）

```
var a = 123;
b = a + ""; //加上空串
console.log(typeof b);
console.log(b);
```

```
var c = 123;
c = c + "";
console.log("c = " + c);
//  输出"c = 123"
```

```
var result;
result = 1 + 2 + "3";
console.log("result" + result);
//  输出"result = 33"
```

```
var result;
result = "1" + 2 + 3;
console.log("result" + result);
//  输出"result = 123"
```

注意运算的顺序

减法

```
var a = 100;
var result = a - "1";
console.log("result = " + result);
//  输出99
```

乘法

```
var result = 2 * "8";   //输出16
var result = 2 * null;  //输出NaN
var result = 2 * undefined; //输出0
```

除法

任何值做减法、乘法、除法运算，都会自动转换为Number

利用上面的准则，将其他数据类型转换为Number类型（隐式类型转换）

```
var a = "100";
a = a - 0;
console.log(typeof a);
console.log("a = " + a);
```

```
var a = "1689";
a = a * 1;
console.log(typeof a);
console.log("a = " + a);
```

```
var a = "1689";
a = a / 1;
console.log(typeof a);
console.log("a = " + a);
```

取模

16 一元运算符

> 利用一元运算符进行数值的隐式转换是很方便的，直接一个加号，适用于任何数据类型

typeof( )就是一个一元运算符

```
正号  +
负号  -   //  可以对符号进行取反
```

非Number值在进行取反运算的时候会先转换为Number类型再取反

```
var a = true;
a = -a;
console.log(typeof a);
console.log("a = " + a);
//  输出"a = 0"
```

一元运算符的隐式转换

```
var a = "100";
a = +100;
console.log(typeof a);
console.log("a = " + a);
//  输出"a = 100"且数据类型是Number
```

```
var a = 1 + +"2" + 3;
console.log(typeof a);
console.log("a = " + a);
//  输出类型为Number，a = 6;
```

17 自增和自减运算符

自增 a++ 和 ++a

> 对于原变量是没有区别的，都会立即给原变量增加1，所以如果输出的是原变量a，那么就是2
>
> 不同的是a++和++a的值（这两个表达式是不同的，所以表达式的值也是不同的）
>
> 分清输出的是表达式的值还是新值是判断计算结果的关键

a++的值还是1，还是自增前的值

++a的值是2，是自增后的新值

```
var a = 1;
a++;
console.log("a++ = " + a++);
//  输出"a++ = 1"
```

```
var a = 1;
++a;
console.log("a = " + ++a);
//  输出"++a = 2"
```

```
var c =10;
c++;    //原变量为10，自增后新值为11，表达式值为10
console.log(c++);   //原变量为11，自增后值为12，但表达式的值为11
```

```
var a = 10;
++a;    //原变量为10，自增后新值为11
a++;    //原变量为11，自增后新值为12
++a;    //原变量为12，自增后新值为13
console.log("a++ = " + a++);    //原变量为13，自增后新值为14，但是表达式a++的值为13
console.log("++a = " + ++a);    //原变量为14，自增后新值为15，表达式++a值为15
```

```
var d = 20;
d = d++;
console.log("d = " + d);    //输出"d = 20"
```

自减 a-- 和 --a

理解了自增后不难理解自减

19 逻辑运算符

```
!    //非
&&  //与
||  //或
```

取反运算 ！

对布尔值进行运算

```
var a = true;
a = !a;
console.log("a = " + a);
```

对非布尔值进行运算（先转换为布尔值）

```
var a = 10;
a = !a;
console.log(typeof a);
console.log("a = " + a);
//输出为"a = false"
```

- Boolean类型的隐式转换（为任意的数据类型进行两次取反操作）

```
var a = 10;
a = !!a;
console.log(typeof a);
console.log("a = " + a);
//输出Boolean，"a = true"
```

与运算 &&

> 只要有一个false就返回false
>
> 只有两个true才返回true
>
> 如果第一个是false那就不用再看

```
true && true == true;
true && false == false;
false && false == false;
```

或运算 ||

> 只要有true就返回true

```
false || false == false;
true || false == true;
true || true == true;
```

21 赋值运算符

> 把符号右侧的值赋值给左侧的变量

```
=
a += 5  ->> a = a + 5
a -= 5  ->> a = a - 5
a *= 5  ->> a = a * 5
a /= 5  ->> a = a / 5
a %= 5  ->> a = a % 3
```

22 关系运算符

> 通过关系运算符可以比较两个值之间的大小关系，如果关系成立返回true，如果不成立则返回false
>
> 注意两个字符串比较可能出错的情况怎样解决

```
>
>=
<
<=
```

非数值之间的关系运算

> 对于非数值进行比较时，会将非数值转换为数值再比较
>
> 任何值和NaN比较结果都是false
>
> 如果符号两边的值都是字符串，不会将其转换为数值进行比较，而是会比较其unicode编码值

```
console.log("10" > 5)   //返回true，因为"10"转换为了10大于5为真
console.log("a" < 10);  //返回false，因为非数值"a"转换为了NaN
console.log("a" < "b"); //返回true
```

- 当比较运算符两边都是字符串时，比较的是字符串的字符编码，而且是一位一位顺着进行比较，所以"11111"和"115"比较是前者小于后者，因为比较到第三位的时候1小于5

```
console.log("1111" < "115");    //输出true
console.log("abc" > "bcd"); //输出false
console.log("bbc" > "bb");  //输出true
```

> 可以利用这个特点对英文进行排序，比较中文没有意义

- 怎样解决比较两个字符串型数据时比较结果可能出错的问题

```
var a = "1111115";
var b = "15648";
var result = a < b; //我们想要的结果是false，但是输出true
var result = a < +b;    //这样"b"就转换为数值型15648了，再比较就会一个数值型一个非数值型，这样非数值型就会转换为数值型比较
```

24 相等运算符和不相等运算符

相等运算符

> 当使用==来比较两个值时，如果两个值的类型不同，则会将值的类型进行转换再进行比较，具体转换成什么数据类型不确定，但是大部分是转换为数值

```
==
console.log(1 == 1);
console.log(1 == "1");  //输出true
console.log(true == "1");   //输出true
console.log(true == "hello")    //输出false
//将字符串和布尔值都转换为数值类型，true转换为1，"hello"转换为NaN
console.log(null == 0); //输出false
```

- undefined衍生自null，所以使用相等运算符时返回true

```
console.log(undefined == null); //返回true
```

- NaN不和任何值相等，包括它本身

```
console.log(NaN == NaN);    //返回false
var a = NaN;
console.log(a == NaN);  //返回false
```

- 通过isNaN( )函数判断一个值是否是NaN

```
var a = NaN;
console.log(isNaN(a));  //返回true
```

不相等运算符

> 不相等也会对数据进行类型转换

```
!=
console.log(10 != 5);
console.log("abc" != "abc");
console.log("1" != 1);
```

- 全等运算符===，和相等运算符相似，但是它不会转换数据类型，数据类型不同直接返回false

```
console.log(null === undefined) //返回false
```

- 不全等运算符!==，和不相等运算符相似，但是不会转换数据类型，数据类型不同直接返回true

25 条件运算符（三目运算符）

```
条件表达式?语句1:语句2;
条件为true执行语句1
条件为false执行语句2
```

```
true?alert("语句1"):alert("语句2");
false?alert("语句1"):alert("语句2");
```

```
var a = 10;
var b = 20;
a > b? alert("a大"):alert("b大");
```

- 获取a和b中的最大值

```
var max = a > b ? a : b;
```

- 获取a和b和c中的最大值

```
var max = a > b ? a : b;
max = max > c ? max : c;
```

26 运算符的优先级

> 和数学中一样，在JS中也有运算符的优先级，先乘除后加减
>
> 优先级一样从左往右运算
>
> JS运算优先级表
>
> 使用括号改变运算的优先级

```
var result = 1 || 2 && 3;
```

```
var result = (1 || 2) && 3;
var result = 1 || (2 && 3);
```

28 if语句

- 注意if条件语句的判断区间不要重叠
- JS键盘输入的函数prompt("Please input")，函数会返回输入的值，需要用变量接收
- prompt( )函数的返回值类型是String类型的

```
var input = prompt("Please input");
```

```
var num1 = +prompt("Please input number.");//将输入的值转换为number类型
```

33 条件分支表达式

```
switch(条件表达式){
    case 1:
        console.log("1");
        break;
    case 2:
        console.log("2");
        break;
    case 3:
        console.log("3");
        break;
    ...
    ...
    default:
        console.log("rest situation");
        break;
}
```

使用if和switch都可以，互相可以实现对方的功能。

46 对象的介绍

区别与其他5种基本数据类型的引用数据类型Object

基本数据类型都是单一的值，值和值之间没有联系，这种数据类型是有缺陷的

对象属于一种复合数据类型，包含保存了多个基本数据类型

- 对象的分类
    - 内建对象：ES标准定义的对象
    - 宿主对象：由JS运行环境提供的对象，目前主要指由浏览器提供的对象
    - 自定义对象：由开发人员自己定义的对象

47 对象的基本操作

- 创建对象（new一个对象）

```
var obj = new Object();
console.log(obj);
```

- 向对象添加属性

```
var obj = new Object();
obj.name = "Jeffery";
obj.gender = "male";
obj.age = 24;
console.log(obj);
```

- 读取对象中的属性

```
console.log(obj.name);
console.log(obj.gender);
console.log(obj.age);
console.log(obj.hobby); //不会报错，返回undefined
```

- 修改对象中的属性值

```
var obj = new Object();
obj.name = "Jeffery";
obj.name = "Jacky";
```

- 删除对象中的属性值

```
var obj = new Object();
obj.name = "Jeffery";
obj.gender = "male";
obj.age = 24;
delete obj.age;
```

48 属性名和属性值

> 使用[ ]的方式添加属性和属性值，这种方式会很灵活，不会把属性写死，可以通过将属性名设置为变量名去添加

```
var obj = new Object();
var n = "changeable";
obj[n] = "你好";
console.log(obj[n])
```

JS对象属性值可以是任意的数据类型，也可以是另一个对象（套娃）

```
var obj1 = new Object();
obj1.name1 = "nobody";
var obj2 = new Object();
obj2.name2 = obj1;
console.log(obj2.name1.name2);
```

- 检查对象中是否有某个属性的操作：in 运算符

```
var obj = new Object();
obj.name = "Jeffery";
obj.gender = "male";
var test1 = "name" in obj;  //返回true
var test2 = "gebder" in obj;    //返回true
var test3 = "age" in obj;   //返回false
```

49 基本数据类型和引用数据类型

> 基本数据类型和引用数据类型的内存存储情况

- JS中的变量都是保存在栈内存中的
  - 基本数据类型的**值**直接在栈内存中存储
    - 值与值之间是独立存在的，修改一个变量的值不会影响其他变量
  - 对象是保存在堆内存中的，每创建一个新的对象，就会在堆内存中开辟一个新的空间
    - 对象的变量保存在栈区的不是值，而是对象的内存**地址**（对象的引用），如果两个变量保存的是同一个对象的引用，那么指向的内存地址也是相同的，当一个对象的变量改变时，另一个对象的变量也会受影响而改变

```
var a = 10;
var b = a;
a++;
console.log(a); //返回11
console.log(b); //返回10
```

```
var obj1 = new Object();
obj1.age1 = 24;
var obj2 = obj1;
obj1.age += 1;
console.log(obj1.age1); //返回25
console.log(obj2.age2); //返回25
```

```
var obj1 = new Object();
obj1.name = "Jeffery";
var obj2 = obj1;
obj2 = null;        //obj2的值改变了，不能通过改变的值的地址去访问对象了，所以不会影响
obj1
console.log(obj1);  //返回的是Jeffery
console.log(obj2);  //返回的是null
```

- 当比较两个基本数据类型的值时，就是值和值比较
- 当比较两个引用数据类型时，是比较对象的内存地址
  - 如果两个对象是一模一样的，但是地址不同，也会返回false

```
var obj1 = new Object();
var obj2 = new Object();
obj1.age = 23;
obj2.age = 23;
var test = (obj1 == obj2);
console.log(test);  //返回false
```

50 对象字面量

> 方便的创建对象的方法

- 麻烦的创建对象的方式

```
var obj = new Object();
```

- 使用对象字面量创建对象

```
var obj = {};
obj.name = "Jeffery";
obj.gender = "male";
```

- 使用对象字面量可以在创建对象时直接指定对象的属性

```
var obj = {      name:"Jeffery",
                gender:"male",
                age:23
            };
console.log(obj.name);
console.log(obj.gender);
console.log(obj.age);
```

51 函数的简介

函数也是一个对象，这个对象可以封装一些功能，在需要时使用这些代码

- 函数声明的方式创建函数

```
function fun(){
    var def = "This is a function.";
    console.log(def);
}
fun();  //函数调用
```

- 给一个变量进行赋值，把创建的函数赋值给变量

```
var fun = function(){
    console.log("This is function expression.");
}
```

52 函数的参数

- 定义一个求和的函数，在函数声明的时候指定形参，在函数调用的时候指定实参

```
function sum(a,b){
    console.log("a + b = "+ a+b);
}
sum(10,20);
```

- 在调用函数时，解析器不会对传递的参数的数据类型进行检查，要注意传递的参数非法的问题
- 实参的数目不够的时候会返回undefined

- 实参可以是任意的数据类型

53 函数的返回值

```
function sum(a,b,c){
    var result = a + b + c;
    return result;
}
var receiver = sum(1,2,3);
console.log(receiver);
```

在函数中，return语句后面的语句都不会执行

return可以返回任意类型的值

54 实参可以是任何类型的值

- 实参是对象

```
var obj = {
    name:"Jeffery";
    gender:"male";
    age:23;
    hobby:Music;
};
function fun(o){
    console.log(o.name);
    console.log(o.gender);
    console.log(o.age);
    console.log(o.hobby);
}
fun(obj);
```

- 实参是函数

```
function fun1(a,b){
    return a + b;
}
fun1(10,20);

function fun2(c){
    return c*c;
}
fun2(fun1);
```

- 开发中经常用到的情况：将一个匿名函数作为实参传入函数
- 实参是函数的返回值

```
function fun1(a,b){
    return a + b;
}
fun1(10,20);

function fun2(c){
    return c*c;
}
fun2(fun1(20,30));
```

56 立即执行函数

函数定义后立即被执行，立即执行函数往往只会执行一次

```
(function(){
    alert("我是一个匿名函数");
})();
```

```
(function(){
    console.log("a = "+ a);
    console.log("b = "+b);
})(123,456);
```

57 方法

> 函数作为一个对象的属性称为该对象的方法，调用函数称为调用该对象的方法

```
var obj = new Object();
obj.name = "Jeffery";
obj.age1 = 12;
obj.age2 = 15;
obj.ageAdd = function(){
    return obj.age1 + obj.age2;
}
obj.ageAdd();//调用函数
```

- 枚举对象中的属性（使用for...in语句枚举对象中的属性）

```
//使用for...in语句枚举对象中的属性
//每次执行会将对象中的一个属性赋值给变量n
var obj = {
    name = "Jeffery";
    age = 23;
    gender: "male";
};
for(var n in obj){
    console.log("属性名:"+n);
    console.log("属性值:"+obj[n]);
}
```

58 作用域

作用域值一个变量作用的范围

JS中的作用域分为两种：全局作用域和局部（函数）作用域

全局作用域

> 直接编写在script标签中的JS代码，都在全局作用域
>
> 函数中不使用var创建者声明的变量也会成为全局变量
>
> 全局作用域在页面打开时创建，在页面关闭时销毁
>
> 在全局作用域中有一个全局对象window（代表浏览器的窗口，由浏览器创建），我们可以直接使用
>
> 在全局作用域中，创建的变量都会作为window对象的属性保存
>
> 在全局作用域中，创建的函数都会作为window对象的方法保存

```
var a = 10; //这个变量保存在window对象里了，作为window对象的属性，10作为a属性的值
window.a;
```

```
funvtion fun(){
    console.log("hello");
}
window.fun();
```

- 变量的声明提前

```
console,log("a = "+a);//不会报错，因为变量已经提前声明了但是不会赋值，后面的代码执行后不会报错，由于没有赋值，返回undefined
var a = 10;
```

- 函数声明的提前

```
fun();
function fun(){
    function body;
}
```

函数作用域

> 全局作用域里的一个部分
>
> 函数作用域的开始：调用函数创建函数作用域
>
> 函数作用域的销毁：函数调用完作用域就销毁
>
> 每调用一次函数就会创建新的函数作用域

```
//创建一个变量
var a = 10;//a是在全局作用域下的全局变量，所以在函数里是可以访问到的
function fun(){
    console.log("a = "+a);
}
fun();
```

```
funtion fun(){
    var b = 10;//在函数创建一个变量b，它是一个局部变量，在外部是访问不到的
}
func();
console.log("b = "+b);
```

- 从外边可以看里边，从外边不可以看外边
- 就近原则，谁近找谁

```
var a = "全局作用域下的变量a";
function fun(){
    var a = "函数作用域下的变量a";
    console.log(a);
}
fun();
console.log(a);
```

- 在函数作用域下直接找全局作用域下的变量，用window

```
var a = "全局作用域下的变量a";
function fun(){
    console.log(window.a);
}
fun();
```

- 函数作用域下不使用var关键字声明的变量会成为全局变量

```
// var a = 20;
function fun(){
    console.log("a = "+a);//返回undefined
    var a = 10;
}
fun();
console.log("a = "+a);  //返回20
```

```
function fun(){
    console.log("a = "+a);//返回undefined
    a = 10;//相当于window.a
}
fun();
console.log("a = "+a);  //返回10
```