js
ecmascript js标准
dom js操作页面DOM
bom js操作浏览器

js特点
解释型语言: 不需要编译写完可直接运行
动态语言:变量可以保存任意值
基于原型的面向对象

语言本指向就是向计算机发布指令，命令计算机做一些事情

输出
alert(1); 弹框
document.write('hello word') //向body中输出一个内容， document代表当前文档
console.log(123) 控制台输出

js代码书写位置
1. html标签中
<button onclick="alert('hello')">点击</button>
2. 写在超链接href中
点击
上面虽然js可以运行，但是结构和行为耦合，不方便维护
3. script标签中
<script>
    alert(1);
</script>
4. 写在外部js文件中
<script src="./1.js"></script>
script用于引入外部js文件之后，中间写js代码不会起到作用了

// 单行注释
/**/ 多行注释

05
字面量和变量
字面量:不可改变的量1 2 3 'hello'，可以直接使用
变量:可以用来保存字面量，并且变量值可以任意改变，变量更加方便使用x = 1; x就是变量，可以通过变量对字面量进行描述
声明变量 var 关键字
var a
声明和赋值同时进行

标识符: 在js中所有可以由我们自主命名的都可以称为标识符
例如:变量名 函数名 属性名都是标识符
规则

1.只能包含字母 数字 _ $
2.标识符不能以数字开头
3.标识符不能是ES中关键字和保留字
4.一般采用驼峰命名
JS底层保存标识符的时候，采用的是Unicode编码，所以理论上中文也能当作变量名

07 数据类型
数据类型指的就是字面量类型
六种数据类型
String Number Boolean Null(空) Undefined(未定义) Object
基本数据类型String Number Boolean Null(空) Undefined(未定义)
引用数据类型 Object

\" "
\' '
\n 换行符
\t 制表符
\\ \
str = '\\\\\\'; \\\

08 数值
typeof 变量 typeof运算符检查变量类型
Number 类型 整数 浮点数 NaN

Number.MAX_VALUE js中的最大值
Number.MIN_VALUE js中可以表示的最小的浮点数
如果数字的值超过了最大值 会返回Infinity Infinity是一个字面量
NaN 是一个特殊的数字，Not a Number 表示不是一个数字

如果使用js进行浮点数计算，可能会得到一个不精准的数字，所以千万不能使用js进行对精确度比较高的计算

09 Boolean
Boolean值只有两个true false，主要用于逻辑判断

10 null undefined
null这个值专门用来保存空对象 typeof null // object
undefined 声明但是没赋值返回undefined, 如果没申明直接使用，会报错

11 强制类型转换
类型转换主要指：将其他数据类型转换为 String Number Boolean
1. 将其他数据类型转换为String
方式1: 调用被转换类型的toString()方法，该方法不会修改原变量，会将转换结果返回
    ** null undefined这两个值没有toString()，调用会报错
    true -- > 'true'
    [1,2,3,4].toString() --> '1,2,3,4'
    [1,2,[3, [4]]].toString() --> '1,2,3,4'

```
    {a: 1}.toString() --> '[object Object]'
    [].toString() --> ''
    String(function fn(){}) --> "function fn(){}"
    var fn = function(){}
    fn.toString() -->"function(){}"
    function fn1() {}
    fn1.toString() --> "function fn1() {}"
```

方式2: 调用String()函数，并将被转换的数据作为参数传给Sting()函数

** null undefined 可以直接传入函数，返回'null' 'undefined'，不会调用toString()方法，因为他俩就没有这个方法

** 使用String()对数字和boolean进行强制类型转换，实际上调用的就是toString()方法

2.将其他数据类型转换为Number

方式1: 使用Number()函数转换

0. 如果是纯数字字符串，则直接将其转换为数字，忽略开头的全部0
1. 非纯字符串转换为数字，返回NaN
2. 如果字符串是一个空串，或者全是空格的串，则转换为0

```
Number('') // 0
Number('    ')  // 0
```

3. true --> 1 false --> 0
4. null --> 0
5. undefined --> NaN
6. Number({a: 1}) Number({}) --> NaN
7. Number([]) --> 0 Number([1]) --> 1 Number([1, 2]) --> NaN

方式2: 使用ParseInt() 把字符串转换为整数 ParseFloat字符串转换为浮点数

1. parseInt()可以将一个字符串中的有效的整数内容取出来，然后转换为Number
2. 只要不是数字开头的都返回NaN
4. parseInt('') parseInt(' ') parseInt(true) parseInt(false) parseInt([]) parseInt({}) --> NaN
5. parseInt是专门针对字符串处理的函数，所以如果不是字符串类型，会先把参数转换为字符串类型，然后再进行操作，转换为字符串过程中，如果没有toString()，那就调用String()

parseInt([1,2,3]) --> 1 因为[1,2,3]会调用toString()转换为'1,2,3',然后parseInt('1,2,3')，转换为1

3.其他进制转换10进制

0x 16进制

0o / 0 8进制

0b 2进制

1.不论什么进制，在进行输出的时候，都会转换为10进制输出

2.在ie8一下 070会被认为是8进制，所以可以使用ParseInt('070', 10) // 第二个参数表示当前数字的进制数

4.其他数据类型转换为boolean

0/NaN/0.0 '' undefined null 都会被转成false

非0 ' ' [] {} 都会被转成true

12 运算符也叫操作符

运算符可以对一个或者多个值进行运算，并获取运算结果
typeof 就是一运算符 可以用来获得一个值的类型 返回值是一个字符串
var res = typeof '1';

算术运算符 + - * / %
当对非Number类型的值进行计算的时候，会将这些值转换为Number类型，然后进行计算
NaN和任何值进行运算，结果都是NaN

特例
+ 只要想家的两个数有一个不是数字，都会先将参数转换为字符串，然后拼接
true+'1' --> true1
str+'' 可以利用这一个特点将任意数据类型转为string

任何值做 - * / 运算的时候，都会自动转换为Number
可以利用这一特性进行隐式类型转换
可以通过一个值 -0 *1 /1将其转换为Number

13 一元运算符，只需要一个操作数
+ -
** 对非number类型的，先转换为number，再进行计算
** 如果将其他类型转换为数字，可以直接使用一元+来将其转换为Number，转换原理和Number()一样
** 优先级高于二元运算符
+
不会对数字产生任何影响
-
可以对数字进行取反

14 自增 自减
自增
1.通过自增可以使变量在自身的基础上增加1
2.对于变量自增以后，原变量的值会立即加1
++ --
** 左右必须是一个变量，不能是一个字面量
** 在进行运算的时候，会调用Number()进行转换
```
var d = 20
d++ + ++d + d
20 + 22 + 22 //
var a = 20;
a = a ++
// var e = a++;
// a = e;
```

15 逻辑运算符 || && !
! 用来对一个值
** 对非boolean值进行取反，现调用Boolean()将非boolean值变成boolean值，可以利用这个特点将一个非boolean值转换为boolean值
    !!a 可以将a转换为boolean值

&&/|| 可以对符号两侧的值分别调用Boolean()，返回值进行运算，并返回原值

&& 短路运算
var a = 1 && 2;   两个值都是true，返回后面一个
a // 2
var a = 0 && 2; 返回0
var a = 2 && 0; 返回2
** 如果第一个值是true,则返回第二个原值
** 如果第一个值是false,则直接返回第一个原值

||
var a = 0 || 1; 返回 1
var a = 1 || 0; 返回 1
** 如果第一个值是true, 则返回第一个值
** 如果第一个是false，则返回第二个值

16 赋值运算符
+= -= *= /= %=

17 关系运算符: 比较两个值之间的大小关系，成立返回true, 不成立返回false

> >= < <=
** 非数组进行比较时，调用Number()，双方转为数值，进行比较
** 任何值和NaN进行比较，都返回false
** 特例：符号两边双方都是字符串，这时候对比每个位置的ASCII码的值 1 A a

==
** 有一个是数字，则都调用Number()转换为数字
** Number和字符串比，都调用Number()转换为数字，然后进行比较。
   例：true == [2] false
   true == [1] true
** null == 0 // false
** undefined == 0 // false
** [] == 0 true 数字和数字相比 都转换为数字

isNaN()判断NaN

18 条件运算符(单元运算符)

19 运算符的优先级
, 逗好运算符
    **申明多个变量时候可以使用

20 模式
工厂模式： 使用工厂方式创建的对象类型都是Object，这样就导致无法区分多种类型的对象
function Animal() {

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

构造函数模式
解决工厂模式不能显示类的问题，每次执行构造函数，构造函数内部的方法依会被重复创建

将函数定义在全局作用域中，污、染了全局作用域空间

** 检查一个对象中是否有某一个属性
'a' in obj
使用in检查对象上是否有某一个属性的时候，如果对象中没有但是原型中有，也会返回true
obj.hasOwnProperty('a')
检查对象自身是否具有某一个属性，原型链上的不算

21 垃圾回收
垃圾积攒过多，会导致程序过乱
当一个对象没有任何的变量或属性对他进行引用，此时我们将永远无法操作对象，这种对象就是一个垃圾，这种对象过多会占用大量的内存空间，导致程序运行变慢，这种垃圾必须进行清理
我们需要做的就是将不再使用的对象设置为null

22 内建对象 宿主对象 自定义对象

22.1 array

arr.length 获取数组长度
arr[10].length 11
arr.length = 10; 初始化数组长度为10
arr = [1,2,3,4]
arr.length = 2; // [1,2] 会剪切数组，可以用于修改数组
new Array(3) [,,]
new Array(1,2,3,4) [1,2,3,4]
Array.of(3) [3]

arr.push(i,y,k) 在数组末尾添加一个或者多个元素，并返回数组新的长度
arr.pop() 删除数组最后一个元素，并返回刚刚删除的元素
arr.unshift(i, y, k) 在数组开头添加一个或者多个元素，并返回数组新的长度
arr.shift() 删除数组第一个元素，并返回刚刚删除的元素
arr.forEach((value, index, arr) => { // IE 8 })
arr.slice(start, end)   // 从某个已有的数组，返回选定区域的元素组成的数组，元素组不变
arr.splice(start, length, insert1, insert2) 开始位置，删除个数，添加元素，修改元素组
arr.concat(arr1, arr2) 连接数组，返回一个新的数组，不会修改元素组
arr.join(split) 合并数组元素，返回字符串，原数组不变
arr.reverse() 反转数组 返回反转之后的数组，并且会修改元素组
arr.sort() 数组元素按照字符串顺序排序，影响原数组

```
arr = [1, 2, 3]
arr.sort(function(a, b) {
    // 1 2
    // 2 3
    return a - b;    // 返回值大于0 元素交换位置
                     // 返回值小于等于0  元素位置不变
})
```

22.2 Function
call apply
arguments 是一个类数组
arguments instanceof Array
arguments.callee // 返回当前正在执行的函数对象

22.3 Date
let d = new Date(); // 初始化当前时间
let d = new Date('2012/1/2')
d.getTime() // 获取时间戳
Date.now() // 获取当前时间时间戳 代码执行某一个刻的时间

22.4 Math 不是一个构造函数，属于一个工具类对象
Math.abs(n) // 返回一个数的绝对值
Math.ceil(n) // 对数进行上舍入
Math.ceil(1.1) // 2
Math.floor(n) // 对数进行向下取整
Math.floor(1.9) // 1
Math.round(n) // 四舍五入取整
Math.random() // 生成0 1 之间的随机数 不会出现0 1
Math.round(Math.random() * 10) 0到10之间的随机数
Math.round(Math.random() * (m -n) + n)
Math.max(n1,n2,n3) //多个数中的最大值
Math.min(n1,n2,n3) //多个数中的最小值

22.5 js提供三个包装类，可以将基本数据类型转换为对象
String() 将基本类型字符串转换为字符串对象
Number() 将基本类型数字转换为数字对象
Boolean() 将基本类型boolean值转换为boolean对象
转换之后，具备对象性能，可以具有一些对象的性能，比如添加属性

22.6 字符串
在底层自字符串是以字符串数组的形式保存的
str.length 获取字符串长度
str.charAt(n) 返回字符串指定位置的字符
str.charCodeAt(n) 返回指定位置字符的unicode编码
String.fromCharCode(code) 根据字符编码获取字符
str.concat('asd') 拼接字符串，相当于+，不会对原字符串产生影响
str.indexOf('a') 返回a的下标

str.lastIndexOf('a') 从后向前查询
str.slice(start, end) 返回截取开始和结束之间的字符串，原字符串不变， end可以是负数
str.substring(start, end) 返回截取开始和结束之间的字符串，原字符串不变，end不能是负数，传入负数默认是0，会自动调整开始和结束位置
//str.substr(start, length)
str.split(sp); 分割字符串，并且去除分割符
str.toUpperCase(); 转为大写，并返回
str.toLowerCase(); 转为小写，并返回

22.7 正则表达式
new RegExp(正则表达式, 匹配模式) i g
reg = new RegExp('a',) 匹配只要有a就行
reg.test(str) 符合返回true, 否则返回false

1. | 表示或者
reg = /a|b|c|d/ 匹配a或b或c或d
2. [] 也表示或
reg = [abcd] 匹配a或b或c或d
[a-z] 任意小写字母
[A-Z] 任意大写字母
3. ^ 中括号中^表示除了
reg = [^ab] 除了a和b以外的

4.字符串支持正则表达式的4个方法
str = '1a2b3c4d5f';
arr = str.split(/[a-z]/) 根据任意字母拆解字符串，即使不指定全局匹配，也会全局拆解
index = str.search(/[a-z]/) 搜索字符串中是否含有指定正则表达式匹配内容，返回匹配开始位置的索引，匹配不到返回-1，只会查找第一个，设置全局也没用
result = str.match(/[a-z]/)
根据正则表达式，将符合条件的内容提取出来，默认情况下只会找到第一个符合条件的内容，设置正则表达式为全局匹配，就会返回全部匹配的内容
多个返回结果组成一个数组
result = str.replace(/[a-z]/g, 'ab')
将字符串中指定内容替换为新的内容
默认只会替换第一个

/a{n}/ 通过量词设置正好出现多少次，量词只对前面的紧挨部分起作用
/a{n,m}/ 出现n到m次
/a{n,}/ 出现m次以上

+ 至少出现一次   {1,}
* 0到多个   {0,}
? 0到1个   {0,1}

/^a/ 以a开头
/a$/ 以a结尾
reg = /^a|a$/ 以a开头或者以a结尾

. 任意字符
\ 转译字符
使用构造函数的时候，由于参数是字符串，而\是字符串的转义字符，这时候需要写两个\

\w 匹配任意字母数字_ [0-9a-zA-Z_]
\W 除了字母数组_ [^0-9a-zA-Z_]
\d 匹配数字 [0-9]
\D 除了数字 [^0-9]
\s 匹配空格
\S 除了空格
\b 单词边界
\B 除了单词边界
reg = new RegExp('/\\bchild\\b/')
reg = /\bchild\b/
str = str.replace(/^\s*|\s*$/g, '') 去除开头和结尾的空格

23 宿主对象
DOM 文档对象模型 document object model
文档 document 就是整个HTML
对象 object将网页的每一个部分都转换为对象
模型 model用来描述对象直接的关系，便于获取对象

节点 Node
构成网页的最基本的组成部分，网页中的每一部分都可以称为一个节点
四类
文档节点：整个HTML文档
元素节点：HTML文档中的HTML标签
属性节点：元素的类型
文本节点：HTML标签中的文本内容
this is a param 元素节点
id="pId" 属性节点
this is a param 文本节点

	nodeName	nodeType	nodeValue
文档节点	#document	9	null
元素节点	标签名	1	null
属性节点	属性名	2	属性值
文本节点	#text	3	文本内容

浏览器已经为我们提供了文档节点对象，这个对象是window的属性。就是document可以在页面中直接使用，文档节点代表整个网页

24事件

文档和浏览器窗口中发生的一些特定的交互瞬间
document.onmousemove事件就是有作用的冒泡
event.cancelBubble = true;

事件委派
将事件统一绑定给元素的共同祖先元素，这样当后台元素的事件触发时，会一直冒泡到祖先元素，从而通过祖先元素的响应函数来处理事件
事件委派利用事件冒泡，通过委派可以减少事件绑定的次数，提高程序的性能

```
oUl.onclick = function(event) {
    console.log(event) //当前点击的元素
    const e = event || window.event;
    if(e.target.className == 'link') {
        console.log(1111);
    }
}
```

事件绑定

```
oUl.onclick = function(event) {
    console.log(1)
}
oUl.onclick = function(event) {
    console.log(2)
}
// 2 会出现覆盖
```

obj.addEventListener('click', fn, false) 可以同时给一个对象绑定多个事件，按照绑定的先后顺序执行
obj.attachEvent('onclick', fn) 不同的是先绑定后执行

W3C将事件分为3个阶段
1.捕获阶段
    在捕获阶段时从外层的祖先元素，向目标元素进行事件的捕获，但是默认此时不会触发事件
2.目标阶段
    事件捕获到目标元素，捕获结束开始在目标元素上触发事件
3.冒泡阶段
    事件从目标元素向他的祖先元素传递，一次触发祖先元素上的事件

如果希望在捕获阶段就触发事件，可以将obj.addEventListener('click', fn, true);
IE8及以下只支持冒泡，不支持捕获

25 navigate
BOM 可以是我们通过JS来操作浏览器
BOM对象
    Window 代表整个浏览器窗口，同时window也是网页中的全局对象
    Navigator 代表当前浏览器的信息，通过改对象可以识别不同的浏览器
    Location 代表当前浏览器地址栏信息，通过location可以获取地址栏信息，或者操作浏览器跳转页面

History 代表浏览器的历史记录，可以通过该对象来操作浏览器的历史记录，不能操作具体的历史记录，只能操作浏览器的向前向后翻页，并且只在当次访问有效

Screen 代表用户的屏幕信息，通过该对象可以获取到用户的显示器的相关的信息

ie11 window.navigator.userAgent不能判断是ie了
chrome firefox mesie(ie11不能判断了)

window.ActiveXObject 可以用来判断是不是IE10以下， ie11或转换为true
'ActiveXObject' in window 可以用来判断所有ie, 包括ie11

History
history.length 获取浏览器记录中页面长度
history.back() 用开回退到上一个页面，作用和浏览器的回退按钮一样
history.forward() 可以跳转到下一个页面，作用和前进按钮一样
history.go() 可以用来跳转到执行的页面1表示向前跳转一个页面，-1表示向后跳转一个

Location 分装了地址栏的信息
location.reload(true) 重新加载当前页面, 强制清空缓存，刷新页面
location.replace(path) 使用新的页面替换当前页面，不会保存在浏览器的历史记录中