JavaScript

## 前言

### 编程语言和标记语言的区别

**编程语言**有很强的逻辑和行为能力在编程语言中有很多具有逻辑性和行为能力的指令，这是主动的

**标记语言（HTML）**不用于向计算机发出指令，常用于格式化和链接，标记语言的存在是用来被读取的，是被动的

### 计算机基础

#### 数据存储单位

位（bit）：1bit可以保存一个0或一个1（最小的存储单位）

字节（Byte）：1B=8b

1KB=1024B 1MB=1024KB 1GB=1024MB 1TB=1024GB

#### 浏览器执行js简介

浏览器分为两部分，渲染引擎和js引擎

渲染引擎：用来解析HTML和css，俗称内核，如Chrome的blink，老版本的webkit

Js引擎：也叫js解释器，用来读取网页中的js代码，对其处理后运行，如Chrome的v8

浏览器本身并不会执行js代码，而是通过内置的js引擎来执行，js引擎执行代码时逐行解释每一句源码（转换为机器语言）然后由计算机执行

### JavaScript

JavaScript是世界上最流行的语言之一，是**运行在客户端的脚本语言**

脚本语言：不需要编译，运行过程中由js解释器（js引擎）逐行解释并执行

注释：

单行注释：“//” 快捷键：Ctrl+’/’

多行注释：“/\*\*/” 默认快捷键：Ctrl+alt+A

## 变量

### 声明变量

Var 变量名;

Var 变量名 = 值;

Var 变量名 = 值, 变量名 = 值, 变量名 = 值;

特殊情况：

1. 只声明不赋值，console.log(变量名)结果是undefined
2. 不声明不赋值直接用会报错
3. **不声明直接赋值可以使用，会成为全局变量**

定义一个变量，但没有赋值，值为undefined；未定义，未赋值直接输出会报错

### 变量命名规范

1、由字母（A-Za-z）、数字（0-9）下划线（\_）美元符号（$）组成

2、严格区分大小写

3、不能以数字开头

4、不能是关键字，保留字

5、变量名必须有意义

6、遵守驼峰命名法，首字母小写，后面的单词的首字母大写

7、推荐翻译网站：有道，爱词霸

**注：尽量不要使用‘name’作为变量名**

## 数据类型

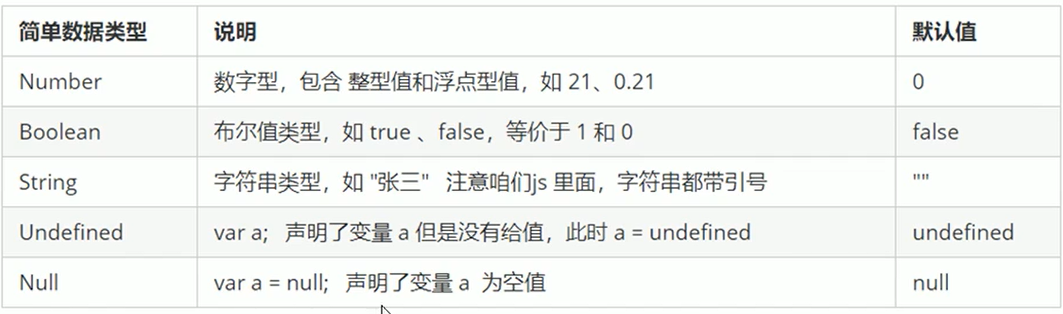
### 变量的数据类型

变量是用来存储值的所在处，他们有名字和数据类型，变量的数据类型决定了如何将代表这些值的位存储到计算机的内存中，JavaScript是一种弱类型或者说动态语言。这意味着不用提前声明变量的类型，在程序运行过程中，类型会被自动确定。

在代码运行时，变量的数据类型是由js引擎根据=右面变量值的数据类型确定的。

**Js是动态语言，var x =10后，可以再将x赋值为其他类型**

### 数据类型之简单数据类型



#### 数字型

进制：Js中八进制前加0，十六进制前加0x

数字型的最大值：Number.MAX\_VALUE，最小值同理

三个特殊值：

Infinity，代表无穷大，大于任何数值

-Infinity，代表无穷小，小于任何数值

NaN，not a number，代表一个非数值

isNaN()方法：用来判断非数字并且返回一个值，若是数字，返回的是false，如果不是数字，返回的是true。

#### 字符串型

字符串型可以使引号中的任意文本其语法为双引号””和单引号’’

因为HTML标签里属性使用的是双引号，js这里更推荐使用单引号

**字符串转义**



length属性：字符串长度。字符串名.length

字符串拼接“+”：字符串+任何类型=拼接后的新字符串

#### Undefined和Null

Undefined+字符串=字符串

Undefined+数字=“NaN”

Null+1=1

## 获取变量类型

typeof 变量名

### 字面量

字面量是在源代码中一个固定值的表示法，通俗来说就是字面量表示如何表达这个值。

数字字面量：7,8,2

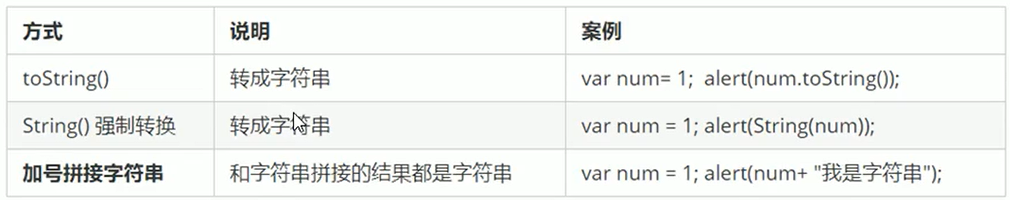
字符串字面量：“黑马”，‘前端’

布尔字面量：true，false

## 数据类型转换

一种数据类型的变量转换成另外一种

### 转换为字符串类型



toString用法：变量.toString

String用法：String(变量名)

### 转换为数字型（重点）



parseInt得到的是整数，只把整数部分提取出来，而且会把单位去掉如120px会变成120（数字必须在开头位置）

‘123’-‘120’结果是3

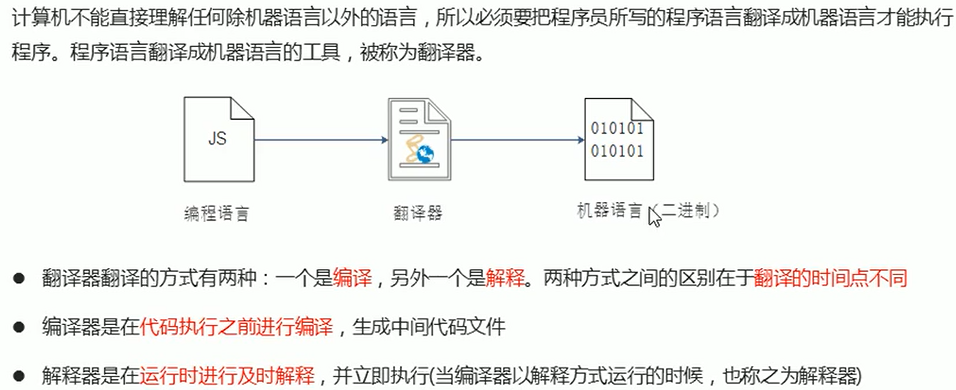
### 转换为布尔型

Boolean()函数

代表空、否定的值会被转换为false，如’’,0,NaN,null,undefined

其余值都会转换为true

## 解释型语言和编译型语言



## 运算符



==默认转换数据类型，会把字符串型转换为数字型，只要值相等就会为true，如37==’37’为true



### 逻辑运算符

#### 短路运算（逻辑中断）

短路运算原理：当有多个表达式（值）时，左边的表达式值可以确定结果时就不再继续运算右边的表达式值。

1. 逻辑与

语法：表达式1 && 表达式2

若第一个表达式值为真，则返回表达式2；若第一个表达式值为假，则返回表达式1

1. 逻辑或（重要，会影响程序运行结果）

若第一个表达式值为真，则返回表达式1；若第一个表达式值为假，则返回表达式2

## 运算符优先级



一元运算符里的逻辑非优先级很高

逻辑与比逻辑或优先级高

## 流程控制（顺序，分支，循环）

### 分支流程控制

#### If语句

If(条件表达式){

//执行语句1

}else{

//执行语句2}

#### 三元表达式

语法结构：条件表达式？表达式1：表达式2

若条件表达式为真，返回表达式1，若为假返回表达式2

#### Switch语句

Switch(表达式){

case value1:

执行语句1;

Break;

case value2:

执行语句2;

Break;

。。。

default:

执行最后的语句;

}

表达式值和case后的选项值匹配，如果匹配得上则执行该case里的语句。

**注：**

1. **开发时，表达式常写为变量**
2. **Switch括号里的变量和case里的值匹配时候是全等，即必须是值和数据类型一致才可以匹配。**
3. **Break如果当前的case里面没有break，则不会退出switch，是继续执行下一个case**

#### If语句和switch的区别

Switch用来比较固定值，if语句用来判断范围；

switch语句进行条件判断后直接执行到程序的条件语句，效率更高。而if…else有几种条件就要判断多少次；

分支比较少时，if语句的效率比switch高；分支多时，switch效率高，结构清晰

## 数组

### 概念

是指一组数据的集合，其中每个数据被称为元素，在数组中**可以存放任意类型的元素。**数组是一种将一组数据存储在单个变量名下的优雅方式

### 数组的使用

访问不存在的索引号输出结果是undefined

增加数组元素方法：

1. 修改length长度
2. 修改索引号，追加数组元素

## 函数

### 声明函数

1、function 函数名(){

//函数体

}

声明函数的关键字全部小写；函数名一般是动词

2、函数表达式（匿名函数）

var 变量名 = function(){};

调用方式：变量名();

函数表达式可以进行传递参数

### 函数的参数

形参和实参个数不匹配问题：

实参个数多于形参：会取到形参的个数；实参个数少于形参则多出的形参可以看做没有声明的变量，结果是undefined；尽量让二者相等

### 函数的返回值

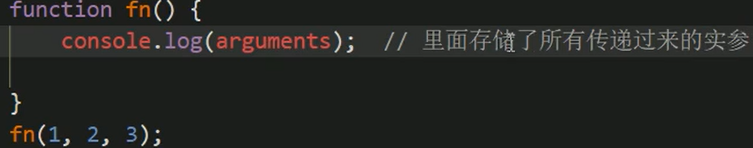
1. return可以终止函数
2. return只能返回一个值，若填多个则返回最后一个
3. 函数若没有return则返回undefined；有return返回其后的值

小结：

Break是结束当前循环体（如for，while）；continue是跳出本次循环，执行下次循环（如for，while）；return不仅可以退出循环还能返回return语句中的值，还可以结束当前的函数体中的代码

### Arguments的使用

当不确定有多少个参数传递的时候，可以用arguments来获取，它是当前函数的一个内置对象，所有函数都内置了一个arguments对象，此对象中存储了传递的所有实参。



arguments展示形式是一个伪数组，可以进行遍历，伪数组的特点为：

具有length属性

按索引方式存储数据

不具有数组的push，pop等方法

## 作用域

Js的作用域（es6之前）：全局作用域 局部作用域

### 变量作用域

全局变量：全局作用下；在函数内部没有直接声明赋值的变量也属于全局变量

局部变量：在局部作用域下，只能在函数内部使用

二者区别：

全局变量只有浏览器关闭的时候才会销毁，比较占内存资源

局部变量在程序执行完完毕就会销毁，比较节约内存资源

Js在es6之前没有块级作用域，在es6的时候新增了块级作用域（if{}，for{}）

## 预解析（笔面试）

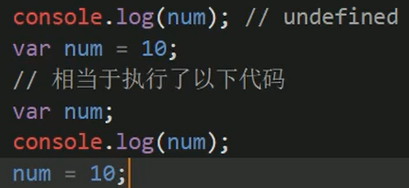
Js引擎运行js分为两步，预解析和代码执行。

预解析：js引擎会把js里面所有的var和function提升到当前作用域的最前面。

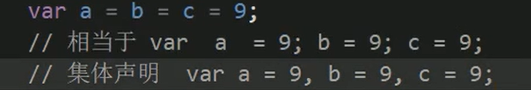
代码执行：按照代码书写的吮吸从上往下依次执行。

预解析分为变量预解析（变量提升）和函数预解析（函数提升）

变量提升就是把所有的变量声明提升到当前的作用域最前面，不提升赋值操作（注，函数表达式调用必须写在函数表达式下面）



函数提升就是把所有的函数声明提升到当前作用域的最前面，不调用函数。



## 对象

### 概念

对象是一组无序的相关属性和方法的集合。

对象由属性和方法组成。属性：事物的特征，在对象中用属性表示，常用名词；方法：事物的行为，在对象中用方法表示，常用动词。

在JavaScript中，对象分为三种，自定义对象，内置对象和浏览器对象，前两种是JS基础内容，属于ECMAscript，第三个浏览器对象属于js独有。

如何查文档学会使用内置对象：通过MDN（推荐）/W3C

### 创建对象的三种方式

#### 利用字面量

对象字面量：就是花括号{}，里面包含了此具体事物（对象）的属性和方法



注：属性或方法采用键值对形式；

多个属性或方法中间用逗号隔开

方法冒号后面跟的是一个匿名函数

调用对象的属性：对象名.属性名 或 对象名[‘属性名’]

调用对象的方法：对象名.方法名()

属性和变量的区别：

变量是单独声明并赋值，使用的时候直接写变量名 单独存在

属性在对象里面不需要声明，使用的时候必须是对象.属性

函数和方法的相同点：都是实现某种功能

不同：函数单独声明并且调用 函数名() 单独存在的

#### 利用new object

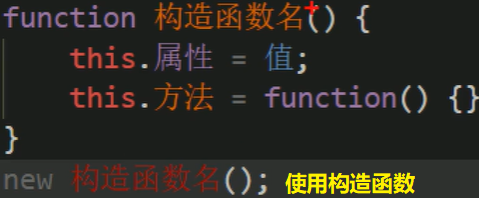


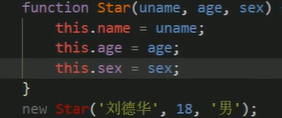
利用等号赋值的方法添加对象的属性和方法；

每个属性和方法之间用分号结束

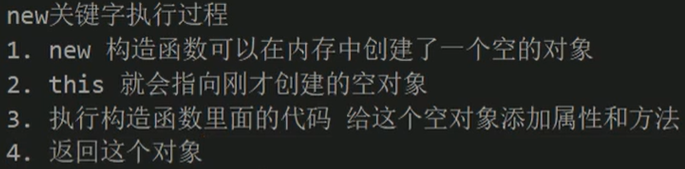
#### 利用构造函数

前两种创建对象的方式一次只能创建一个对象，构造函数就是把我们对象里面一些相同的属性和方法抽象出来封装到函数里面





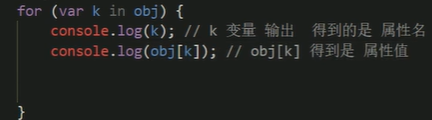
注：构造函数名字的首字母要大写；构造函数不需要return就可以返回结果；



### 遍历对象属性

for…in用于对数组或者对象的属性进行循环操作。

for ( 变量 in 对象){}



### JavaScript的内置对象

#### Math对象

不是一个函数对象，不是一个构造器，Math的所有属性和方法都是静态的，可以直接使用，不需要new调用

##### Math.max()

如果没有参数则结果为-Infinity；如果有任一参数不能被转换为数值，则结果为NaN

##### Math.floor()向下取整

##### Math.ceil()向上取整

##### Math.round()四舍五入就近取整

.5往大了取

##### Math.abs()

隐式转换会把字符串型转换为数值型

##### Math.random()

返回一个随机小数，范围是0≤x＜1，里面无参数

若要得到两个数之间的随机整数并且包含这两个整数：

Math.floor(Math.random() \* (max-min+1))+min;

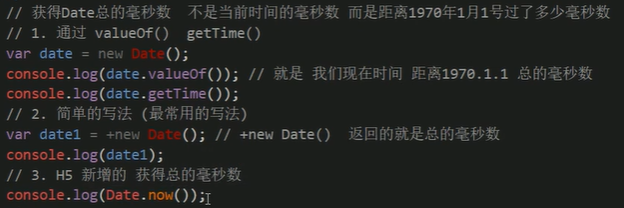
#### Date()对象

只能通过构造函数实例化对象，Date对象没有字面量格式

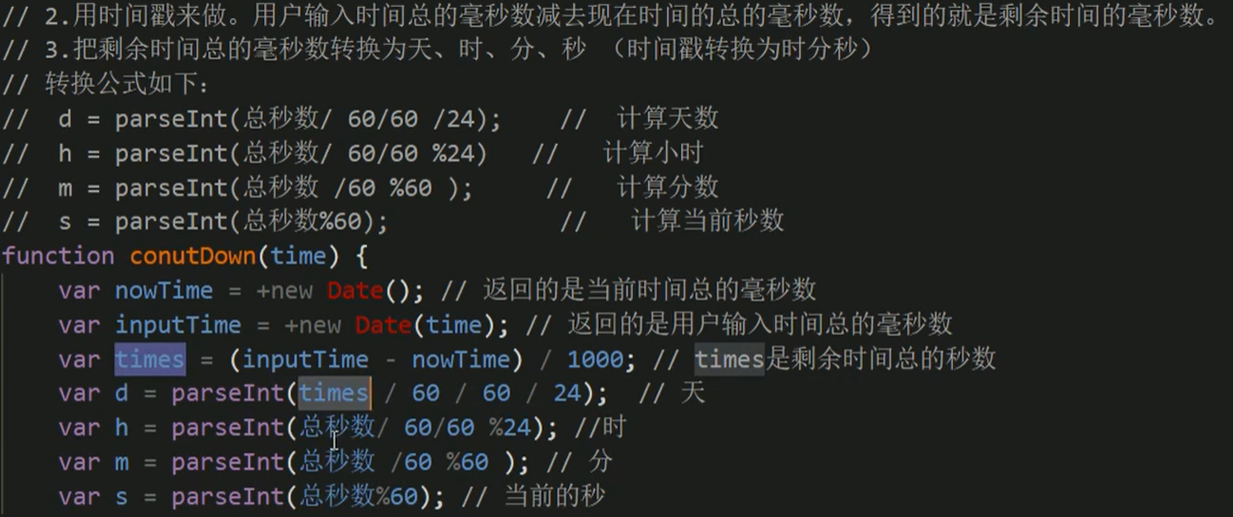
如果Date()没有里参数，返回系统的当前时间



周一返回1，周六返回6，周日返回0



倒计时：



#### 数组对象



##### 检测是否为数组

1. instanceof运算符检测
2. Array.isArray(参数);【注：H5新增方法，IE9以上版本支持】

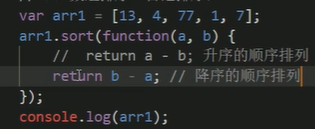
##### 添加删除数组元素



##### 数组排序



Sort排序是按照位数（第一位，第二位等）进行排序，在某些时候会出现问题，解决办法：



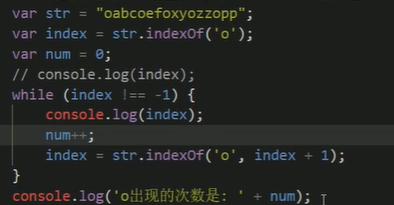
##### 数组索引方法



使用方法：数组名.indexOf(查找元素)

indexOf(‘要查找的字符’,[起始位置])

查找某个元素出现的次数



##### 数组转换为字符串

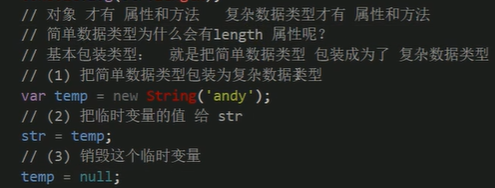


数组名.join()方式转换为字符串时，不写分隔符默认是逗号分隔



#### 字符串对象

基本包装类型，把简单数据类型包装成为了复杂数据类型，使简单数据类型看起来拥有了属性和方法



##### 字符串的不可变性

指的是里面的值不可变，虽然看上去可以改变内容，但其实是地址变了，在内存中新开辟了一个内存空间

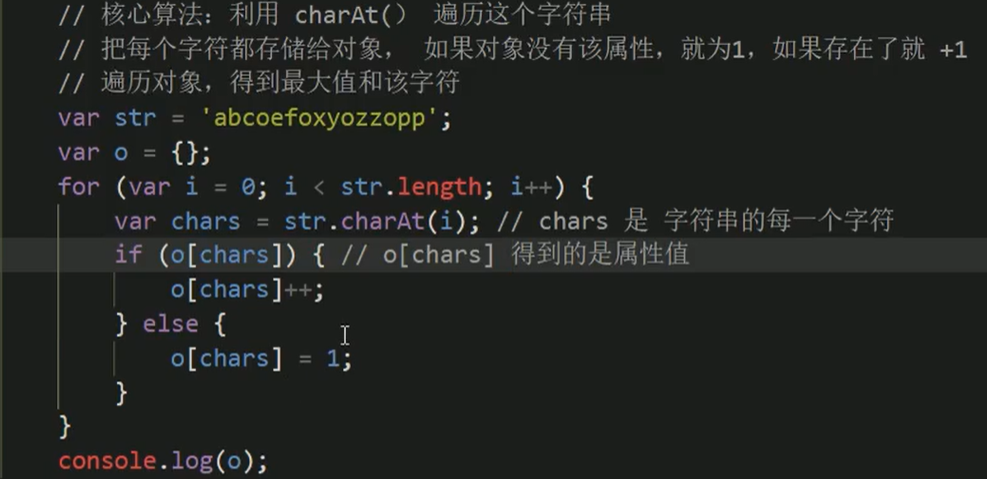
所以字符串所有的方法都不会修改字符串本身（字符串是不可变的），操作完会返回一个新的字符串

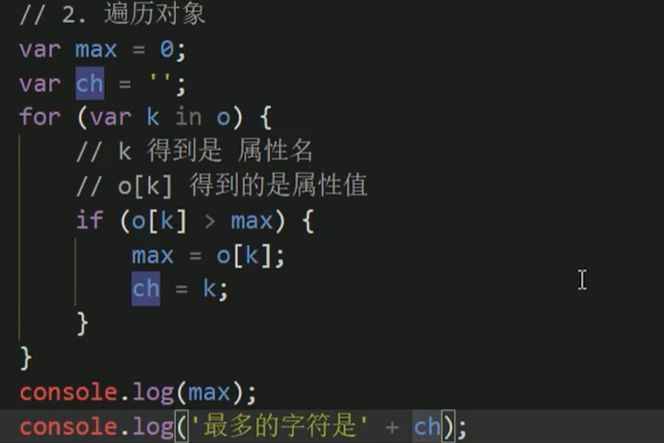
##### 根据位置返回字符



charCodeAt(index)返回相应索引号的字符ASCII值：判断用户按下了哪个键

应用：统计出现最多的字符和次数





##### 字符串操作方法（重点）



## 简单类型和复杂类型

简单类型又叫基本数据类型或者值类型，复杂数据类型又叫引用类型。

值类型：简单数据类型在存储时变量中存储的是值本身，因此叫值类型。

string，number，boolean，undefined，null简单数据类型null返回的是一个空对象（object），如果有变量以后想存为对象，但是没想好放什么，就给null

引用类型：复杂数据类型，在存储时变量中存储的仅仅是地址（引用），因此叫引用数据类型，通过new创建的对象

### 堆和栈

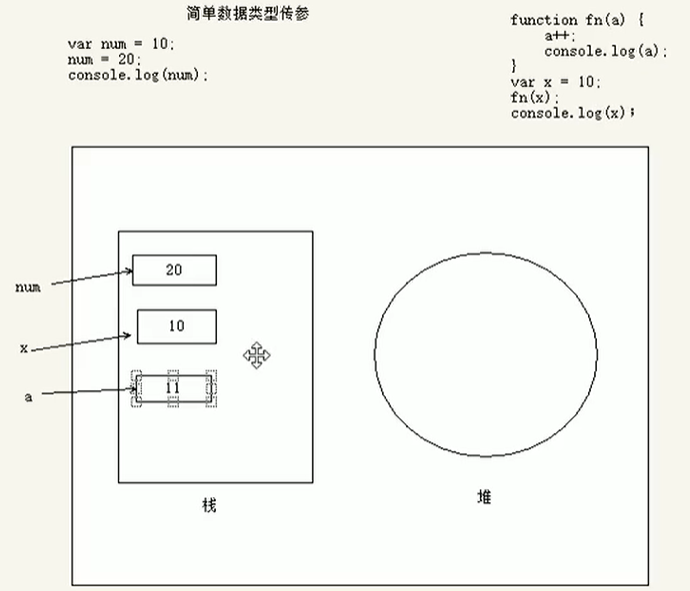
区别：

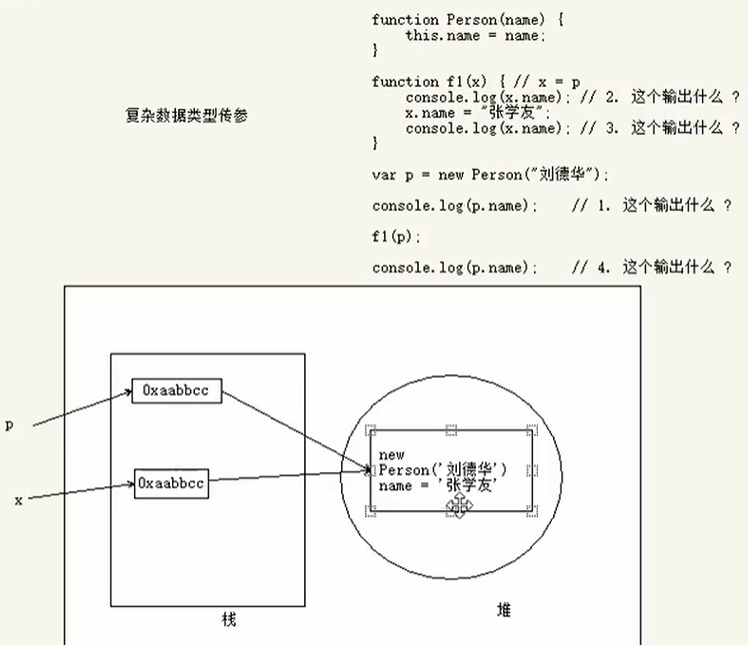
1. 栈：由OS自动分配释放存放函数的参数值，局部变量的值等，其操作方式类似于数据结构中的栈；**简单数据类型放在栈里，直接开辟一个空间存放值**
2. 堆：存储复杂类型（对象），一般有程序员分配释放若程序员不释放，由垃圾回收机制回收。**复杂类型存放到堆里，值放在堆里，栈中存放保存该值的堆的地址**

注意：js中没有堆栈的概念，通过堆栈的方式可以让大家更容易理解代码的一些执行方式，便于将来学习其他语言

### 传参

简单类型传参传的是值；复杂类型传参传的是地址。





## Web APIs

### API和web API

API是给程序员提供的一种工具，以便能更轻松的实现想要完成的功能

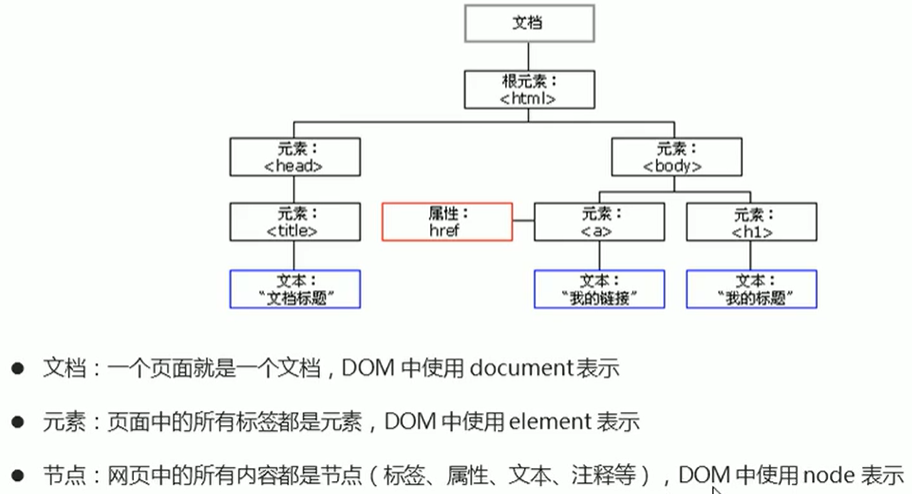
Web API是浏览器提供的一套操作浏览器功能和页面元素的API（BOM和DOM）

### DOM

#### 定义

文档对象模型（Document Object Model，简称DOM），是W3C组织推荐的处理可扩展标记语言（HTML或XML）的标准编程接口

#### DOM树



DOM把以上内容都看作是对象

#### 获取元素

##### 根据ID获取：

使用getElementByld()方法可以获取带有ID的元素对象

语法：var element = document.getElementById(*id*);

**element**是一个 [Element](https://developer.mozilla.org/zh-CN/docs/Web/API/Element) 对象。如果当前文档中拥有特定ID的元素不存在则返回null.

**id**是**大小写敏感的字符串**，代表了所要查找的元素的唯一ID.

返回一个匹配到 ID 的 DOM [**Element**](https://developer.mozilla.org/zh-CN/docs/Web/API/Element)**对象**。若在当前 [Document](https://developer.mozilla.org/zh-CN/docs/Web/API/Document) 下没有找到，则返回 null。

console.dir()打印返回的元素对象，更好的查看里面的属性和方法

<https://developer.mozilla.org/zh-CN/docs/Web/API/Document/getElementById>

##### 根据标签名获取：

使用getElementsByTagName()方法可以返回带有指定标签名的对象的集合

document. getElementsByTagName()可以得到整个文档中的某标签的对象集合；element. getElementsByTagName()可以得到元素里的某标签的对象集合

##### 通过HTML5新增方法获取：

1. document.getElementsByClassName(‘类名’)根据类名返回元素对象集合，不用加符号
2. document.querySelector(‘选择器’)根据指定选择器返回第一个元素对象，切记里面的选择器要加符号
3. document.querySelectorAll()返回指定选择器的所有元素对象集合

##### 获取特殊元素（body，HTML）

获取body元素：var bodyEle = document.body;

获取HTML元素：var htmlEle = document.documentElement;

#### 事件基础

##### 概念

触发—响应机制，由三部分组成，事件源，事件类型，事件处理程序，也称为事件三要素。

事件源：事件被触发的对象

事件类型：如何触发，比如鼠标点击（onclick）鼠标经过，键盘按下等

事件处理程序：通过一个函数赋值的方式完成

##### 执行事件的步骤

1. 获取事件源
2. 注册事件（绑定事件）
3. 添加事件处理

常见的鼠标事件：



#### 操作元素

Js的DOM操作可以改变网页内容，结构和样式，我们可以利用DOM操作元素来改变元素里面的内容属性等。

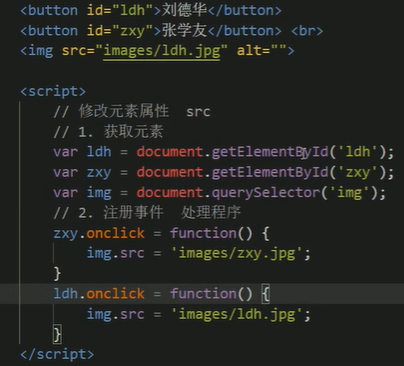
##### 改变元素里的内容

element.innerText从起始位置到终止位置的内容，但它不识别HTML标签，同时空格和换行也会去掉

element.innerHTML起始位置到终止位置的全部内容，包括HTML标签，同时保留空格和换行

这两个属性是可读写的，可以获取元素里面的内容，如console.log(p.innerText);

##### 常用元素的属性操作



##### 表单元素的属性操作

获得焦点：onfocus

失去焦点：onblur

鼠标经过：onmouseover

鼠标离开：onmouseout

##### 样式属性操作

如果修改的比较少可以用element.style进行行内样式操作

如果样式修改的比较多可以用element.className(驼峰命名法)进行类名样式操作

注意1：

js里的样式采用驼峰命名法，如fontSize

JS修改style样式操作产生的是行内样式，权重比较高

**注意2**：

1. 如果样式修改较多，可以采取操作类名方式更改元素样式。

2. class因为是个保留字，因此使用className来操作元素类名属性

3. className 会直接更改元素的类名，会覆盖原先的类名。

##### 自定义属性操作

**获取属性值**

* element.属性 获取属性值。
* element.getAttribute('属性');

**区别**

* element.属性 获取内置属性值（元素本身自带的属性）
* element.getAttribute(‘属性’); 主要获得自定义的属性 （标准） 我们程序员自定义的属性

**设置属性值**

* element.属性 = ‘值’ 设置内置属性值。
* element.setAttribute('属性', '值');

**区别**

* element.属性 设置内置属性值
* element.setAttribute(‘属性’); 主要设置自定义的属性 （标准）

**移除属性值**

* element.removeAttribute('属性');

##### H5自定义属性

H5规定自定义属性data- 开头做为属性名并且赋值。

比如 <div data-index=“1”></div>

或者使用 JS 设置

element.setAttribute(‘data-index’, 2)

**获取H5自定义属性**

1. 兼容性获取 element.getAttribute(‘data-index’);
2. H5新增 **element.dataset.**index 或者 element.dataset[‘index’] ie 11才开始支持

**即：element.dataset. + ‘名字’或者 element.dataset[‘名字’]**

**对于名字太长的自定义属性，在使用第二种方法获取时，名字要使用驼峰命名法，如data-list-name需要变成listName**

H5新增的获取自定义属性方法只能获取data-开头的自定义属性

#### 节点操作

##### 节点概述

一般地，节点至少拥有nodeType（节点类型）、nodeName（节点名称）和nodeValue（节点值）这三个基本属性。

* 元素节点 nodeType 为 1
* 属性节点 nodeType 为 2
* 文本节点 nodeType 为 3 （文本节点包含文字、空格、换行等）

我们在实际开发中，节点操作主要操作的是**元素节点**

##### 节点层级

###### 父节点操作（node.parentNode）

* parentNode 属性可返回某节点的父节点，注意是最近的一个父节点
* 如果指定的节点没有父节点则返回 null

###### 子节点操作（parentNode.childNodes / parentNode.children）

返回包含指定节点的子节点（包含元素节点，文本节点等）的**集合**，该集合为即时更新的集合

**注意：**返回值里面包含了所有的子节点，包括元素节点，文本节点等。

如果只想要获得里面的元素节点，则需要专门处理。 所以我们一般不提倡使用childNodes

**parentNode.children是一个只读属性，返回所有子元素节点，它只返回子元素节点，其余节点不返回**

parentNode.firstChild返回第一个子节点，找不到则返回null。同样也是包含所有的节点（其中有文本节点和元素节点等）。parentNode.lastChild同理

parentNode.firstElementChild返回第一个子元素节点。

parentNode.lastElementChild返回最后一个子元素节点，找不到则返回null。

注：这两个方法有兼容性问题，IE9以上才支持

**实际开发**的写法，既没有兼容性问题又返回第一个子元素：

**parentNode.children[i]**

**解决方案：**

1. 如果想要第一个子元素节点，可以使用 parentNode.chilren[0]
2. 如果想要最后一个子元素节点，可以使用 parentNode.chilren[parentNode.chilren.length - 1]

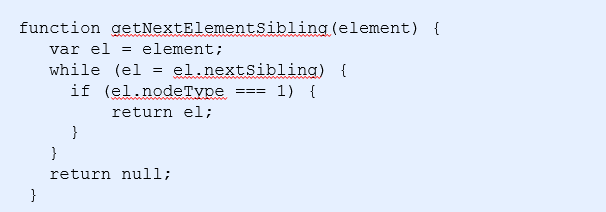
###### 兄弟节点（nextSibling）

node.nextSibling是下一个兄弟节点，包含元素节点或者文本节点，找不到则返归null

node.previousSibling是返回元素上一个兄弟节点，找不到则返回null，同样包含所有节点

nextElementSibling得到下一个兄弟元素节点，previousElementSibling得到上一个兄弟元素节点，但是有兼容性问题

解决办法：

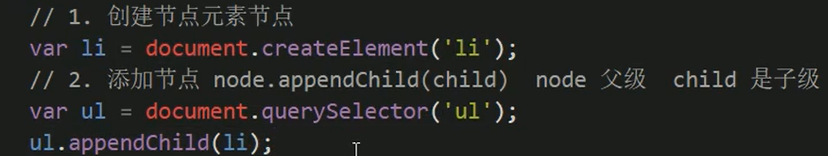


###### 创建节点（document.createElement(‘tagName’)）

document.createElement() 方法创建由 tagName 指定的 HTML 元素。因为这些元素原先不存在，是根据我们的需求动态生成的，所以我们也称为动态创建元素节点。

###### 添加节点(node.appendChild(child))

1、在创建节点元素节点后添加，node.appendChild(child)其中node是父级，child是子级，追加方式添加，即添加到父节点的子节点列表的末尾



2、node.insertBefore()方法将一个节点添加到父节点的指定子节点前面，类似于before伪元素

用法：node.insertBefore(child, 指定元素)

###### 删除节点

node.removeChild();

**<a href="javascript:;"></a>可以阻止链接跳转**

###### 复制节点

node.cloneNode()方法返回调用该方法的节点的一个副本，也称为克隆节点。其中node是被克隆的节点

**注意：**

1. 如果括号参数为空或者为 false ，则是浅拷贝，即只克隆复制节点本身，不克隆里面的子节点。

2. 如果括号参数为 true ，则是深度拷贝，会复制节点本身以及里面所有的子节点。

##### 三种动态创建元素的区别

document.write()

用法：document.write(‘<div>1222</div>’);

element.innerHTML

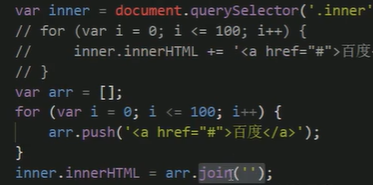
document.createElement()

**区别：**

1. document.write 是直接将内容写入页面的内容流，但是**文档流执行完毕，则它会导致页面全部重绘**

2. innerHTML 是将内容写入某个 DOM 节点，不会导致页面全部重绘

3. innerHTML 创建多个元素效率更高（不要拼接字符串，采取数组形式拼接），结构稍微复杂

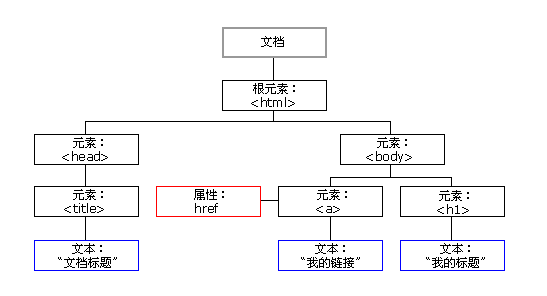


4. createElement() 创建多个元素效率稍低一点点，但是结构更清晰

**总结：不同浏览器下，innerHTML不采用字符串拼接方式使用数组时 效率要比 creatElement 高**

问题二

##### DOM重点核心（小结）



我们获取过来的DOM元素是一个对象（object），所以称为 文档对象模型

1. 创建

document.write

innerHTML

createElement

1. 增

appendChild

insertBefore

1. 删

removeChild

1. 改

修改元素属性： src、href、title等

修改普通元素内容： innerHTML 、innerText

修改表单元素： value、type、disabled等

修改元素样式： style、className

1. 查

DOM提供的API 方法： getElementById、getElementsByTagName 古老用法 不太推荐

H5提供的新方法： querySelector、querySelectorAll 提倡

利用节点操作获取元素： 父(parentNode)、子(children)、兄(previousElementSibling、nextElementSibling) 提倡

1. 属性操作

setAttribute：设置dom的属性值

getAttribute：得到dom的属性值

removeAttribute移除属性

1. 事件操作

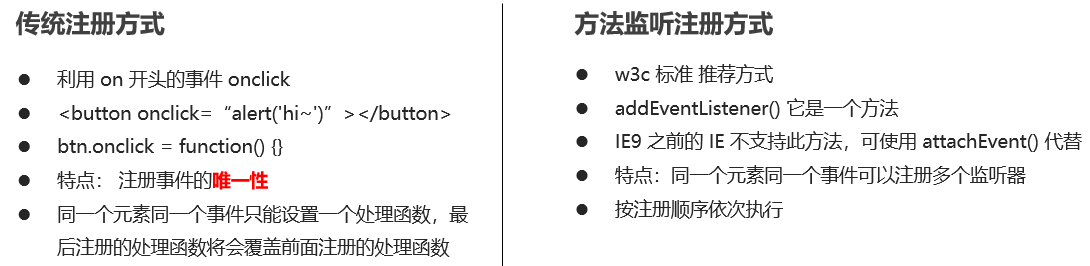


### 事件高级

#### 注册事件（绑定事件）

给元素添加事件，称为注册事件或者绑定事件。

注册事件有两种方式：传统方式和方法监听注册方式



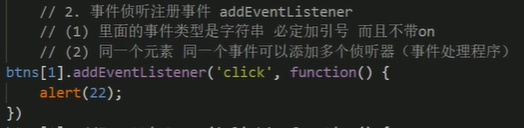
##### addEventListener事件监听方式

eventTarget.addEventListener(type, listener[, useCapture])

eventTarget.addEventListener()方法将指定的监听器注册到 eventTarget（目标对象）上，当该对象触发指定的事件时，就会执行事件处理函数。

该方法接收三个参数：

* type：事件类型字符串，比如 click 、mouseover ，注意这里不要带 on
* listener：事件处理函数，事件发生时，会调用该监听函数
* useCapture：可选参数，是一个布尔值，默认是 false。学完 DOM 事件流后，我们再进一步学习



##### attachEvent事件监听方式（了解）

eventTarget.attachEvent(eventNameWithOn, callback)

eventTarget.attachEvent()方法将指定的监听器注册到 eventTarget（目标对象） 上，当该对象触发指定的事件时，指定的回调函数就会被执行。

该方法接收两个参数：

* eventNameWithOn：事件类型字符串，比如 onclick 、onmouseover ，**这里要带 on**
* callback： 事件处理函数，当目标触发事件时回调函数被调用

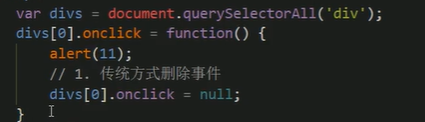
**注意：IE8 及早期版本支持**

##### 注册事件兼容性解决方案



#### 删除事件（解绑事件）

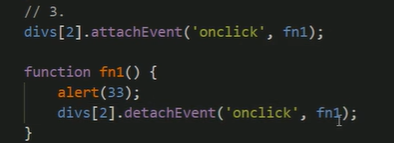
1、传统注册方式：eventTarget.onclick = null;



2、方法监听注册方式

* 1. eventTarget.removeEventListener(type, listener[, useCapture]);
  2. eventTarget.detachEvent(eventNameWithOn, callback);





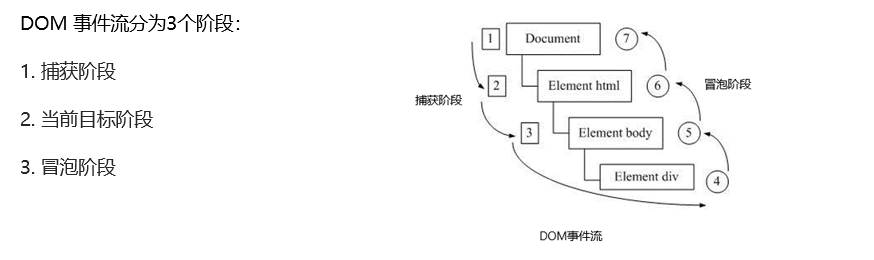
3、删除事件的兼容性解决方案



#### DOM事件流

事件流描述的是从页面中接收事件的顺序。

事件发生时会在元素节点之间按照特定的顺序传播，这个传播过程即 DOM 事件流。比如我们给一个div 注册了点击事件：



* 事件冒泡： IE 最早提出，事件开始时由最具体的元素接收，然后逐级向上传播到到 DOM 最顶层节点的过程。
* 事件捕获： 网景最早提出，由 DOM 最顶层节点开始，然后逐级向下传播到到最具体的元素接收的过程。

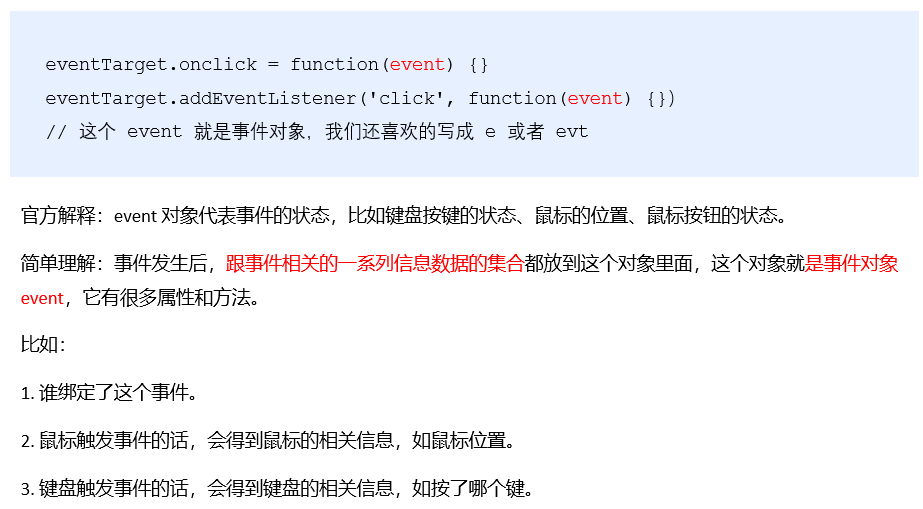
事件发生时会在元素节点之间按照特定的顺序传播，这个传播过程即 DOM 事件流。

**注意**

1. JS 代码中只能执行捕获或者冒泡其中的一个阶段。
2. onclick 和 attachEvent 只能得到冒泡阶段。
3. addEventListener(type, listener[, useCapture])**第三个参数如果是 true，表示在事件捕获阶段调用事件处理程序；如果是 false（不写默认就是false），表示在事件冒泡阶段调用事件处理程序。**
4. 实际开发中我们很少使用事件捕获，我们更关注事件冒泡。
5. 有些事件是没有冒泡的，比如 onblur、onfocus、onmouseenter、onmouseleave
6. 事件冒泡有时候会带来麻烦，有时候又会帮助很巧妙的做某些事件，我们后面讲解。

#### 事件对象（重要）

##### 什么是事件对象



1. event就是一个事件对象，写到侦听函数的小括号里面，当形参来看
2. 事件对象只有有了事件才会存在，是系统自动给我们创建的，不需要传递参数
3. 事件对象是我们事件的一系列相关数据的集合
4. 事件对象可以自己命名，如evt，e，event
5. 有兼容性问题，ie678通过window.event。解决方式：

event=event || window.event;

##### 事件对象的常见属性和方法

e.target 和 this 的区别：

this 是事件绑定的元素， 这个函数的调用者（**绑定这个事件的元素**）

e.target 是**事件触发**的元素。

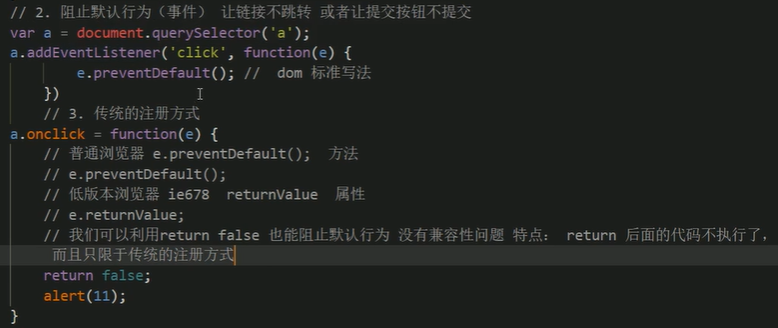
总结：e.target点击了哪个元素，就返回哪个元素this是哪个元素绑定了这个点击事件，那么就返回谁







阻止默认行为：return false;没有兼容性问题，但是后面如果有代码无法执行，而且只限于传统的注册方式onclick



#### 阻止事件冒泡（面试常见）

事件冒泡：开始时由最具体的元素接收，然后逐级向上传播到到 DOM 最顶层节点。

事件冒泡本身的特性，会带来的坏处，也会带来的好处，需要我们灵活掌握。

##### 阻止事件冒泡的两种方式

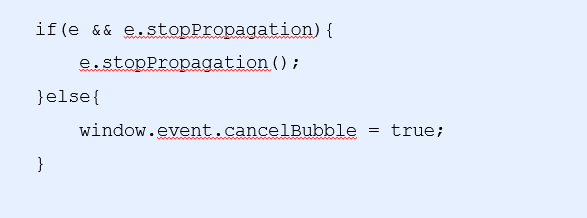
* 非标准写法：IE 6-8 利用事件对象 cancelBubble 属性

e.cancelBubble = true;

* 标准写法：利用事件对象里面的 stopPropagation()方法

e.stopPropagation()

##### 阻止事件冒泡的兼容性解决方案



#### 事件委托

##### 事件委托原理（重要）

**不是每个子节点单独设置事件监听器，而是事件监听器设置在其父节点上，然后利用冒泡原理影响设置每个子节点。**

以上案例：给 ul 注册点击事件，然后利用事件对象的 target 来找到当前点击的 li，因为点击 li，事件会冒泡到 ul 上， ul 有注册事件，就会触发事件监听器。

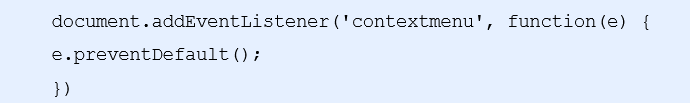
**作用：我们只操作了一次 DOM ，提高了程序的性能。**

#### 常用鼠标事件

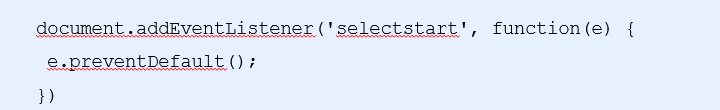


##### 禁止鼠标右键菜单

contextmenu主要控制应该何时显示上下文菜单，主要用于程序员取消默认的上下文菜单



##### 禁止鼠标选中



##### 鼠标事件对象

event对象代表事件的状态，跟事件相关的一系列信息的集合。现阶段我们主要是用鼠标事件对象 MouseEvent 和键盘事件对象 KeyboardEvent。



#### 常用键盘事件

##### 常用键盘事件



**注意：**

**1. 如果使用addEventListener 不需要加 on**

1. onkeypress 和前面2个的区别是，它不识别功能键，比如左右箭头，shift 等。
2. 三个事件的执行顺序是： keydown -- keypress --- keyup

##### 键盘事件对象



**注意： onkeydown / keydown和 onkeyup / keyup 不区分字母大小写，onkeypress 区分字母大小写。**

在我们实际开发中，我们更多的使用keydown和keyup， 它能识别所有的键（包括功能键）

Keypress 不识别功能键，但是**keyCode属性能区分大小写，返回不同的ASCII值**

### BOM浏览器对象模型

#### BOM概述

##### 什么是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。

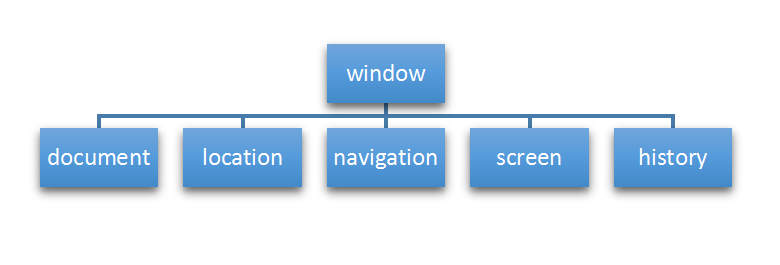
BOM 由一系列相关的对象构成，并且每个对象都提供了很多方法与属性。

BOM 缺乏标准，JavaScript 语法的标准化组织是 ECMA，DOM 的标准化组织是 W3C，BOM 最初是Netscape 浏览器标准的一部分。



##### BOM的构成

BOM比DOM更大，它包含DOM



**window 对象是浏览器的顶级对象，**它具有双重角色。

1. 它是 JS 访问浏览器窗口的一个接口。
2. 它是一个全局对象。定义在全局作用域中的变量、函数都会变成 window 对象的属性和方法。

在调用的时候可以省略 window，前面学习的对话框都属于 window 对象方法，如 alert()、prompt() 等。

**注意：**window下的一个特殊属性 window.name，打印后会输出空，故命名的时候不要使用name

#### Window对象的常见事件

##### 窗口加载事件

window.onload = function(){} 或者 window.addEventListener("load",function(){});

window.onload 是窗口 (页面）加载事件,当文档内容完全加载完成会触发该事件(包括图像、脚本文件、CSS 文件等), 就调用的处理函数。

**注意：**

1. 有了 window.onload 就可以把 JS 代码写到页面元素的上方，因为 onload 是等页面内容全部加载完毕，再去执行处理函数。

2. window.onload **传统注册事件方式 只能写一次**，如果有多个，会以最后一个 window.onload 为准。

3. 如果使用 addEventListener 则没有限制

document.addEventListener('DOMContentLoaded',function(){})

DOMContentLoaded 事件触发时，仅当DOM加载完成，不包括样式表，图片，flash等等。Ie9以上才支持

如果页面的图片很多的话, 从用户访问到onload触发可能需要较长的时间, 交互效果就不能实现，必然影响用户的体验，此时用 DOMContentLoaded 事件比较合适。

注：load等页面内容全部加载完毕，包含页面DOM元素 图片 flash css等；DOMContentLoaded是DOM加载完毕，不包含图片 flash css等就可以执行，加载速度比load快

##### 调整窗口大小事件

window.onresize = function(){}调整窗口大小加载事件，**触发时就调用的处理函数**

window.addEventListener("resize",function(){});

注：我们经常利用这个事件完成响应式布局。 window.innerWidth 当前屏幕的宽度

#### 定时器

##### setTimeout()定时器

window.setTimeout(调用函数,[延迟的毫秒数]);延迟的毫秒数可以省略，默认为0

setTimeout()方法用于设置一个定时器，该定时器在定时器到期后执行调用函数。

**注意：**

1. window 可以省略。

2. 这个调用函数可以直接写函数，或者写函数名或者采取字符串‘函数名()'三种形式。第三种不推荐

3. 延迟的毫秒数省略默认是 0，如果写，必须是毫秒。

4. 因为定时器可能有很多，所以我们经常给定时器赋值一个标识符。

setTimeout() 这个调用函数我们也称为**回调函数 callback**

普通函数是按照代码顺序直接调用。而这个函数，需要等待时间，时间到了才去调用这个函数，因此称为回调函数。

简单理解： 回调，就是回头调用的意思。上一件事干完，再回头再调用这个函数。

以前我们讲的element.onclick = function(){}或 element.addEventListener(“click”, fn); 里面的 函数也是回调函数。

##### 停止setTimeout()定时器

window.clearTimeout(timeoutID)

clearTimeout()方法取消了先前通过调用 setTimeout() 建立的定时器。

**注意：**

1. window 可以省略。

2. 里面的参数就是定时器的标识符 。

##### setInterval()定时器

window.setInterval(回调函数,[间隔的毫秒数]);

setInterval()方法**重复调用一个函数**，每隔这个时间就去调用一次回调函数

注意：

1. window 可以省略。

2. 这个调用函数可以直接写函数，或者写函数名或者采取字符串‘函数名()'三种形式。第三种不推荐

3. 延迟的毫秒数省略默认是 0，如果写，必须是毫秒。

4. 因为定时器可能有很多，所以我们经常给定时器赋值一个标识符。

区别：

setTimeout延时时间到了就去调用这个回调函数，只调用一次就结束这个定时器

setInterval每隔这个延时时间，就去调用这个回调函数，会调用很多次，重复调用这个函数

##### 停止setInterval()定时器

window.clearInterval(intervalID);

clearInterval()方法取消了先前通过调用setInterval()建立的定时器

注意：

1. window可以省略
2. 里面的参数就是定时器的标识符



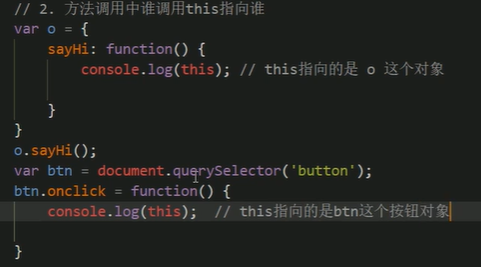
##### this

this的指向在函数定义的时候是确定不了的，只有函数执行的时候才能确定this到底指向谁，一般情况下this的最终指向的是那个调用它的对象

现阶段，我们先了解一下几个this指向

1. 全局作用域或者普通函数中this指向全局对象window（注意定时器里面的this指向window）

2. 方法调用中谁调用this指向 谁



3.构造函数中this指向构造函数的实例

#### JS执行队列

##### JS是单线程

JavaScript 语言的一大特点就是单线程，也就是说，同一个时间只能做一件事。这是因为 Javascript 这门脚本语言诞生的使命所致——JavaScript 是为处理页面中用户的交互，以及操作 DOM 而诞生的。比如我们对某个 DOM 元素进行添加和删除操作，不能同时进行。 应该先进行添加，之后再删除。

单线程就意味着，所有任务需要排队，前一个任务结束，才会执行后一个任务。这样所导致的问题是： 如果 JS 执行的时间过长，这样就会造成页面的渲染不连贯，导致页面渲染加载阻塞的感觉。

##### 同步和异步

为了解决上面提到的问题，利用多核 CPU 的计算能力，HTML5 提出 Web Worker 标准，允许 JavaScript 脚本创建多个线程。于是，JS 中出现了同步和异步。

同步：前一个任务结束后再执行后一个任务，程序的执行顺序与任务的排列顺序是一致的、同步的。比如做饭的同步做法：我们要烧水煮饭，等水开了（10分钟之后），再去切菜，炒菜。

异步：你在做一件事情时，因为这件事情会花费很长时间，在做这件事的同时，你还可以去处理其他事情。比如做饭的异步做法，我们在烧水的同时，利用这10分钟，去切菜，炒菜。

**他们的本质区别： 这条流水线上各个流程的执行顺序不同。**

同步任务：同步任务都在主线程上执行，形成一个**执行栈。**

异步任务：**JS 的异步是通过回调函数实现的**。

一般而言，异步任务有以下三种类型:

1、普通事件，如 click、resize 等

2、资源加载，如 load、error 等

3、定时器，包括 setInterval、setTimeout 等

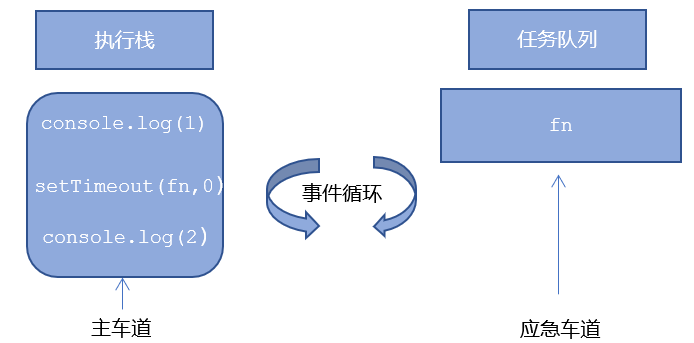
异步任务相关回调函数添加到**任务队列**中（任务队列也称为消息队列）。

##### 执行机制

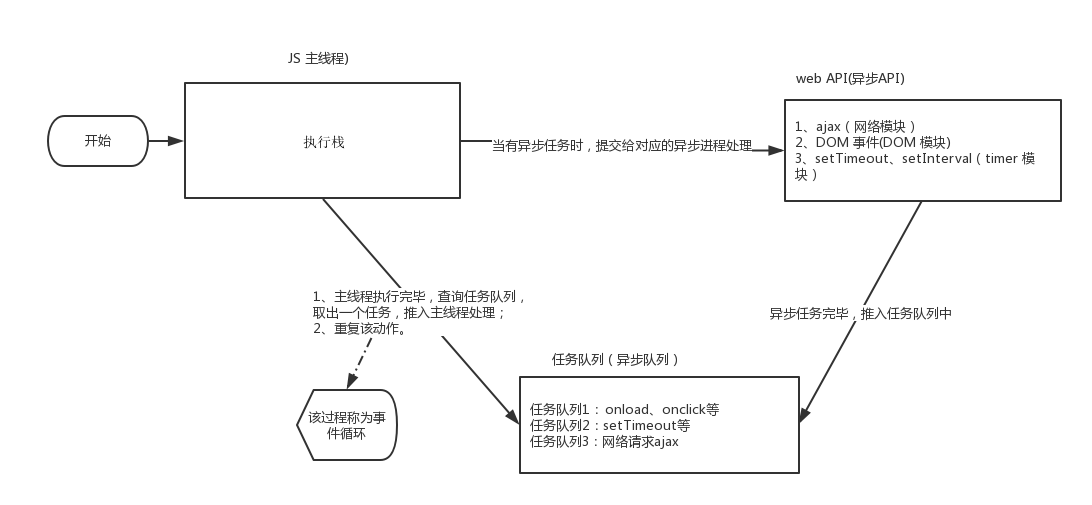
1. 先执行执行栈中的同步任务。

2. 异步任务（回调函数）放入任务队列中。

3. 一旦执行栈中的所有同步任务执行完毕，系统就会按次序读取任务队列中的异步任务，于是被读取的异步任务结束等待状态，进入执行栈，开始执行。



##### 多个异步任务



由于主线程不断的重复获得任务、执行任务、再获取任务、再执行，所以这种机制被称为事件循环（ event loop）。

#### Location对象

##### URL

统一资源定位符 (Uniform Resource Locator, URL) 是互联网上标准资源的地址。互联网上的每个文件都有一个唯一的 URL，它包含的信息指出文件的位置以及浏览器应该怎么处理它。

URL 的一般语法格式为：

protocol://host[:port]/path/[?query]#fragment

http://www.itcast.cn/index.html?name=andy&age=18#link

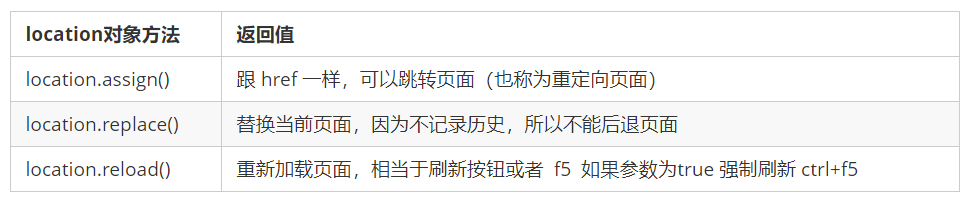


##### Location对象

window 对象给我们提供了一个 location 属性用于获取或设置窗体的 URL，并且可以用于解析 URL 。 因为这个属性返回的是一个对象，所以我们将这个属性也称为 location 对象。



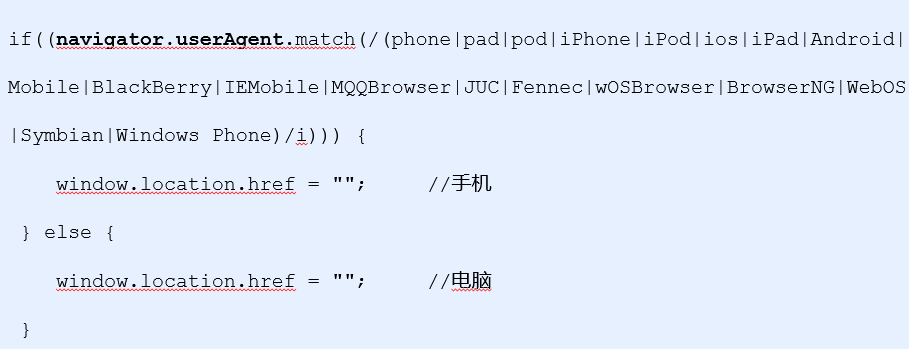
重点记住href和search



#### Navigator对象（了解）

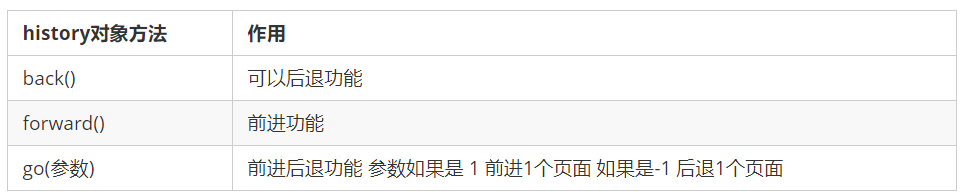
navigator 对象包含有关浏览器的信息，它有很多属性，我们最常用的是 userAgent，该属性可以返回由客户机发送服务器的 user-agent 头部的值。

下面前端代码可以判断用户那个终端打开页面，实现跳转



#### History对象

window 对象给我们提供了一个 history 对象，与浏览器历史记录进行交互。该对象包含用户（在浏览器窗口中）访问过的 URL。



history 对象一般在实际开发中比较少用，但是会在一些 OA 办公系统中见到。

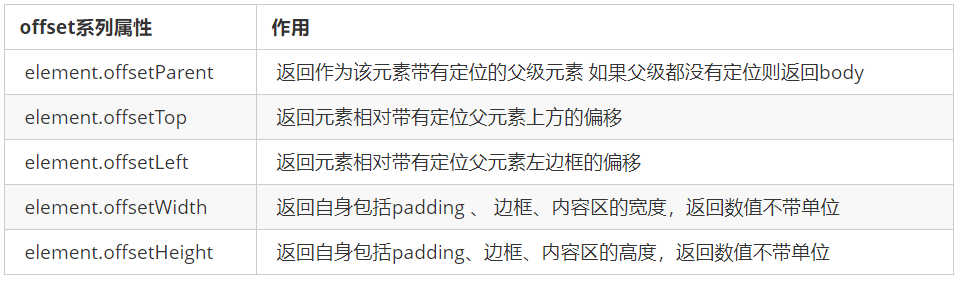
### PC端网页特效

#### 元素偏移量offset系列

##### Offset概述

offset 翻译过来就是偏移量， 我们使用 offset 系列相关属性可以动态的得到该元素的位置（偏移）、大小等。

* 获得元素距离带有定位父元素的位置
* 获得元素自身的大小（宽度高度）
* 注意： 返回的数值都不带单位



Element.offsetParent和element.parentNode区别：后者返回的是最近一级的父亲，不管有没有定位

##### offset与style区别



## 1

### 2

#### 3

##### 4

###### 5