**浏览器的构成**

一般由7个模块组成：User Interface（用户界面）、Browser engine（浏览器引擎）、Rendering engine（渲染引擎）、Networking（网络）、JAVAScript Interpreter（js解释器）、UI Backend（UI后端）、Date Persistence（数据持久化存储）

**浏览器内核**

浏览器内核是浏览器的核心，也称“渲染引擎”，用来解释网页语法并渲染到网页上。浏览器内核决定了浏览器该如何显示网页内容以及页面的格式信息。不同的浏览器内核对网页的语法解释也不同，因此网页开发者需要在不同内核的浏览器中测试网页的渲染效果。

|  |  |
| --- | --- |
| 浏览器内核 | 主要浏览器 |
| Trident | **IE浏览器** |
| webkit | **Safari历览器** |
| Blink | **Chrome浏览器、Opera浏览器** |
| Gecko | **Firefox浏览器** |

**JS的特性**

解释性语言

解释型语言是对代码进行一句一句的直接运行，在程序运行期间，使用解释器动态将代码解释为机器码，再运行。

单线程

JS 是单线程的，但是却能执行异步任务，这主要是因为 JS 中存在事件循环（Event Loop）和任务队列（Task Queue）。

ECMA标准

**JS三大部分**

ECMAScript DOM BOM

**JS引入**

<script>代码</script> 页面级

<script type=“text/javascript” src=“文件名”></script> 外部JS文件

**JS变量**

原始值

Number、Boolean、String、Undefined、Null

原始变量及他们的值储存在栈中

引用值

Array、Object、Function、Date、RegExp

引用变量的名称储存在栈中，但是把其实际对象储存在堆中

**堆和栈**

栈(stack)：栈会自动分配内存空间，会自动释放，存放基本类型，简单的数据段，占据固定大小的空间。

堆(heap)：动态分配的内存，大小不定也不会自动释放，存放引用类型，指那些可能由多个值构成的对象，保存在堆内存中，包含引用类型的变量，实际上保存的不是变量本身，而是指向该对象的指针。

**typeof**

typeof对象、数组或null返回object

typeof(a) ; undefined

typeof(NaN)； number

**JS显式转换和隐式转换**

**显示转换**

Number()

接收一个参数：123，true,false,null,'123'能处理，其他均是NaN

parseInt()

两个参数：对象+进制、123，'123','123a'，其他均为NaN

该进制参数指数是此进制下的 如parseInt(3，3) == NaN

parseFloat()

一个参数：123，'123.11', '123.1a'，其他均为NaN

String()

接收一个参数：接收任何参数相当于+’’

obj.toString()

接收一个参数：进制 -> 可将数转换成此进制下的数

Boolean()

接收一个参数：''、NaN、null、undefined、0是false，其他均为true

**隐式转换**

++、-- (Number) +'123'、-'123'

\*、/、-、%、+ (Number或String)

>、< (Number或String）

**特殊**

undefined == null;

isNaN（obj）==>Number(obj)

if，do-while，while语句会将判断语句转换为布尔类型值。

记：-1+NaN=NaN

**JS运行三步骤**

语法分析

预编译

解释执行

imply global 暗示全局变量：即任何变量，如果变量未经声明就赋值，此变量就为全局对象所有。

一切声明的全局变量，全是window的属性。

**预编译**

预编译发生在代码执行的前一刻

步骤：

1.创建GO/AO对象

AO Activation Object 执行期上下文 函数预编译，在函数执行前一刻

GO Global Object 在整个代码运行前一刻

2.找形参和变量声明，将变量和形参名作为AO属性名，值为undefined

3.将实参值和形参统一

4.在函数体里面找函数声明，值赋予函数体 （注：没有函数表达式 function () {} ）

**运行期上下文**

当js控制器(control)进入可执行代码时，控制器会进入一个执行环境，活动的多个执行环境构成执行环境栈，最上面的是正在运行的执行环境，当控制器进入一个新的执行环境时，新的执行环境被放到栈最顶端。

函数每次执行时对应的执行上下文都是独一无二的，所以多次调用一个函数回导致创建多个执行上下文，当函数执行完毕，它所产生的执行上下文被销毁。

|  |  |
| --- | --- |
| 组件 | 作用目的 |
| 词法环境 | **执行环境内的代码创建的标识符引用的一个词法环境对象** |
| 变量环境 | **执行环境内的代码通过变量表达式和函数表达式创建的绑定的一个词法环境对象** |
| this绑定 | **指定该执行环境内的this关键字所关联的值。** |

**执行环境包含以下三个组件：**

**作用域 [[scope]]**

有些属性仅供JavaScript引擎存取，[[scope]]就是其中一个，指的就是我们所说的作用域，其中存储了执行期上下文的集合。

**作用域链**

[[scope]]中所存储的执行期上下文对象集合，这个集合呈链式链接，我们把这种链式链接叫做作用域链。

查找变量：从作用域链的顶端依次向下查找。

手机屏幕截图

描述已自动生成

手机屏幕的截图

描述已自动生成

**闭包**

闭包是指有权访问另一个函数作用域中的变量的函数。

当内部函数被保存到外部时，将会生成闭包。闭包会导致原有作用域链不释放，造成内存泄漏。

**闭包的作用**

实现公有变量

eg：函数累加器

可以做缓存（存储结构）

eg：eater

可以实现封装，属性私有化。

模块化开发，防止污染全局变量

**立即执行函数**

声明一个函数，并马上调用这个匿名函数就叫做立即执行函数

(function () {} () ); W3C 建议第一种

(function () {} ) ();

只有表达式才能被执行符号执行 -> 执行符号（）

能被执行符号执行的表达式，函数名会被忽略 ->匿名函数

**对象的创建方法**

1、var obj = { …… } plainObject 对象字面量/对象直接量

2、构造函数

1）系统自带的构造函数 new Object ( )

2) 自定义 外部添加属性和对象

3、工厂模式

**包装类**

Boolean、String、Number

原始值不能有属性和方法，当给它们添加属性的时候系统会自动进行包装类并销毁。

**原型**

每个函数都有prototype(原型)属性，这个属性是一个指针，指向一个对象，这个对象的用途是包含特定类型的所有实例共享的属性和方法，即这个原型对象是用来给实例共享属性和方法的。而每个实例内部都有一个指向原型对象的指针。

获取对象原型

console.log(Person.prototype);//构造函数.prototype,获取对象原型

console.log(person1.\_\_proto\_\_);//对象实例.\_\_proto\_\_，获取对象原型

查看构造函数

console.log(person1.constructor);//对象实例.constructor，获取构造函数

绝大多数对象的最终都会继承自Object.prototype

Object.create(原型)；

Object.create(null)；没有原型，什么东西都没，没有\_\_proto\_\_，而且自己加是不行的

\_\_proto\_\_系统给了可以改，系统没给不能添加

**call/apply 传参列表不同**

Person.call(obj, "Jim", 20); //改变了this的指向

call需要把实参按照形参的个数传进去

Person.call(obj, ["Jim", 20]);

apply 需要传一个arguments

**继承**

1、传统形式 --> 原型链

过多的继承了没用的属性

2、借用构造函数

不能继承借用的构造函数的原型

每次构造函数都要多走一个函数

3、共享原型

不能随便改动自己的原型,因为指向同一个空间。

4、圣杯模式（原型链继承）

基于共享原型的基础上增加一个中间层

**命名空间**

管理变量，防止污染全局，适用于模块化开发

**属性使用**

obj.name

obj["name"]

**对象的枚举**

对象属性的遍历

for in -> 用法for (var prop in person) { }

1、hasOwnProperty

person.hasOwnProperty(prop) //返回boolean型，判断是不是属于自己的属性

继承的属性会输出 false

2、in

“LastName”in person//false 判断该对象能不能调用到属性

3、instanceof

A instanceof B //返回boolean型，判断A对象的原型链上有没有B的原型

obj instanceof Object --> Object.prototype是否存在于参数obj的原型链上。

**区分 [ ]数组和{ }对象，本身是不知道的一个变量**

1. 利用constructor

[ ].constructor ->Array { }.constructor ->Object

2、利用 instanceof

[ ] instanceof Array -> true的是数组

{ } instanceof Array -> false的是对象

3、利用toString

Object.prototype.toString.call ( [ ] )； -> Array

Object.prototype.toString.call ( { } )； -> Object

**深度拷贝**

1、判断是不是原始值 typeof()、instanceof、toString、constructor

万无一失用 toString

是原始值就直接拷贝

是引用值进行下一步

2、判断是数组还是对象

把引用值当作新的拷贝对象

3、建立相应的数组或对象

**数组**

1、var arr = new Array(10);只有一个参数，默认为数组长度，不能为浮点数

区别 var arr = [10];第一个元素为10

2、在JS中不存在越界的问题 arr[10] -> undefined

可以溢出写 直接写入超过数组长度的位置也行

3、push进栈、pop出栈、unshift在数组最前面加入元素、reverse逆转元素

**splice截取元素、替换元素**

两个参数a,b，从a位截取b个元素

三个元素a,b,c，把从a位开始的b个元素 替换为c

**sort排序**

arr.sort(compare); compare为一个比较大小的函数 a>b return 1 这种

**slice截取**

一个参数a，从a位到最后全部截取

两个参数a,b，从a位到b位

**join连接**

参数为一个字符串，数组元素之间由参数连接，函数返回字符串

字符串的方法中有split，作用相反

**类数组**

属性要为索引（数字）属性，必须要有length属性，最好加上push **length必须！**

**try……catch**

在try里面发生错误，不会执行错误后的try里面的代码

catch捕获错误信息，并进行操作

**Error.name 六种值对应的信息**

1、EvalErroe：eval() 的使用与定义不一致

2、RangrError： 数值越界

3、ReferenceError：非法或不能识别的引用数值

4、SyntaxError：发生语法解析错误

5、TypeError ：操作数类型错误

6、UriError：URI处理函数使用不当

**DOM**

Document Object Model DOM定义了表示和修改文档所需的方法。DOM对象即为宿主对象，由浏览器厂商定义，用来操作xml和html功能的一类对象的集合。也有人称DOM是对HTML以及XML的标准编程接口

**DOM节点类型**

元素节点 　　Node.ELEMENT\_NODE(1)

属性节点 　　Node.ATTRIBUTE\_NODE(2)

文本节点 　　Node.TEXT\_NODE(3)

CDATA节点 Node.CDATA\_SECTION\_NODE(4)

实体引用名称节点 　　Node.ENTRY\_REFERENCE\_NODE(5)

实体名称节点 　　Node.ENTITY\_NODE(6)

处理指令节点 　　Node.PROCESSING\_INSTRUCTION\_NODE(7)

注释节点 　 Node.COMMENT\_NODE(8)

文档节点 　 Node.DOCUMENT\_NODE(9)

文档类型节点 　　Node.DOCUMENT\_TYPE\_NODE(10)

文档片段节点 　　Node.DOCUMENT\_FRAGMENT\_NODE(11)

DTD声明节点 Node.NOTATION\_NODE(12)

**节点的四大属性**

nodeName

元素的标签名，以大写的形式表示，只读

nodeValue

Text节点或Comment节点的文本内容，可读写

nodeType

该节点的类型，只读

attributes

Element节点的属性集合

**节点的一个方法：Node.hasChildNode();**

如果节点有子节点则返回true ，否则返回false。

**DOM的基本操作**

**1、查、获取**

**查看元素节点**

document代表整个文档

document.getElementById()

//元素id 在Document.prototype上

.getElementsByTagName（）

//标签名 写在了Document Element这两个的原型上

.getElementsByName（）

//需注意，只有部分标签name可生效（表单，表单元素，img，iframe）

// 在 HTMLDocument.prototype上

.getElementsByClassName（）

//类名 -->ie8和ie8以下的ie版本中没有，可以多个class一起

//使用中注意一下，返回类型

.querySelector()

//css选择器 在ie7和ie7以下的版本中没有

.querySelectorAll()

//css选择器 在ie7和ie7以下的版本中没有

.querySelector() .querySelectorAll() 不是实时的，只是一个副本

**遍历节点树**

parentNode -->父节点

（最顶端的parentNode为#document）document没父节点

childNodes -->子节点们

包括文本节点如回车 多个文本内容在一起算一个节点

firstChild -->第一个子节点

lastChild -->最后一个子节点

nextSiling -->后一个兄弟节点

previousSiling -->前一个兄弟节点

**基于元素节点树的遍历**

parentElement -->返回当前元素的父元素节点（IE不兼容）

children -->只返回当前元素的元素子节点

node.childElementCount() === node.child.length当前元素节点个数（IE不兼容）

firstElementChild -->返回第一个元素节点（IE不兼容）

lastElementChild -->返回最后一个元素节点（IE不兼容）

nextElementSiling/previousElementSibling -->只返回后一个/当前一个兄弟节点（IE不兼容）

**2、增**

document.createElement(); -->创建元素节点

document.createTextNode() -->创建文本节点

document.createComment() -->创建注释节点

**3、插**

PARENTNODE.appendChild();　　-->将某一个节点插入到前边的节点里边

//新建的创建类似push，已有的插入就是剪切

PARENTNODE.insertBefore(a,b) -->在某个节点内，将a插入到b前边

**4、删**

parent.removeChild(); -->父元素节点删除某个子节点，算是剪切出来

child.remove(); -->节点自己删除（自杀式删除）

**5、替换**

parent.replaceChild(New,origin); -->用新的节点替换旧的节点

**6、Element节点的一些属性**

innerHTML

字符串类型

在某个元素节点中添加相关的内容，也可以查询

innerText

只有文本内容

只有文本，就直接改了；但是有标签，innerText使用后全部覆盖，包括标签

**7、Element节点的一些方法**

ele.setAribute("属性名称"，“属性值”); -->在某个元素节点中添加某个属性

ele.getAribute("属性名称")　　　　 -->查询某个元素节点的属性

**DOM结构树**

手机屏幕截图

描述已自动生成

HTMLDocument.prototype上定义了一些常用的属性，body，head分别指的html文档中的<body><head>标签

document.documentElement 指的是 html文档

**日期对象Date()**

var date = new Date();date出生时刻，不变了！

getDate()

// 根据本地时间返回指定日期对象的 月份中的第几天（1-31）

getDay()

// 根据本地时间返回指定日期对象的 星期中的第几天（0-6）

getMonth()

// 根据本地时间返回指定日期对象的 月份（0-11）

getFullYear()

// 根据本地时间返回指定日期对象的 年份（四位数年份时 返回四位数字）

getHours()

// 根据本地时间返回指定日期对象的 小时（0-23）

getMilliseconds()

// 根据本地时间返回指定日期对象的 毫秒（0-999）

getMinutes()

// 根据本地时间返回指定日期对象的 分钟（0-59）

getSeconds()

// 根据本地时间返回指定日期对象的 秒数（0-59）

getTime()

// 返回从1970-1-1 08:00:00 UTC（协调世界时）到该日期经过的毫秒数

// 对于 1970-1-1 08:00:00 UTC 之前的时间返回负值

时间戳是指格林威治时间1970年01月01日00时00分00秒(北京时间1970年01月01日08时00分00秒)起至现在的总 秒数。

时间戳可以计算程序运行的时间

**JS定时器**

window.setTimeout(code,millisec);

可以使一段代码在指定时间后运行

window.setInterval(code,millisec);

可以使一段代码每过指定时间就运行一次

millisec设置后不可更改

//此函数非常不准

有返回值，唯一标识符

定时器清除的方法：clearTimeout(obj)和clearInterval(obj)

# 手机屏幕截图 描述已自动生成JS获取滚动条的高度

**让滚动条滚动**

window上有三个方法

scroll( )，scrollTo( ) 两个功能是一样的

scrollBy( ); 会在之前的数据基础上做累加

三个方法功能类似, 用法都是将x, y坐标传入. 即实现让滚动条滚动到当前位置

区别: scrollBy( )

可以用在展开收起这种功能上，收起后要回到未展开是的位置，就利用滚动条的功能

**查看视口的尺寸**

window.innerWidth/innerHeight

IE8及IE8以下不兼容

document.documentElement.clientWidth/clientHeight

标准模式下, 任意浏览器都兼容

document.body.clientWidth/clientHeight

适用于怪异模式下的浏览器 （渲染模式）

**查看元素的几何尺寸**

.getBoundingClientRect( )； 兼容性很好

该方法返回一个对象, 对象里面有left, top, right, bottom等属性。left和top代表该元素左上角的x和y坐标，right和bottom代表元素右下角的x和y坐标

height和width属性老版本IE并未实现

返回的结果并不是“实时的”

**查看元素的尺寸**

dom.offsetWidth, dom.offsetHeight

视觉上的尺寸，包括padding

**查看元素的位置**

dom.offsetLeft, dom.offsetTop

对于无定位父级的元素, 返回相对于文档的坐标. 对于有定位父级的元素, 返回相对于最近的有定位的父级的坐标.

dom.offsetParent

返回最近的有定位的父级, 如无, 返回body, body.offsetParent返回null

eg: 求元素相对于文档的坐标getElementPosition( )

**脚本化CSS**

DOM可以间接操作CSS

每一个dom元素都有一个属性style，dom.style里面存放的这个元素的行间样式，我们可以通过这个属性来读写元素的行间样式。

注意：

1.我们碰到float这样的关键字属性的时候，前面要加一个css前缀。

float—>cssFloat

2.复合属性必须拆解

border: 1px solid red;

这种属性是不允许出现的，只能写border-width、border-color、border-style。

3.当css中的样式是用“-”连接起来的时候，我们要变成小驼峰式写法。

background-color —> backgroundColor;

这个属性只能读写行间样式，我们写在css的样式是不能获取的，因此我们通过这个属性加样式的时候要注意权重值问题。

同时，我们在js中写样式只能往行间写，永远也写不到css文件里面。

**查询计算样式**

window上面有一个方法叫做getComputedStyle可以来获取元素的计算样式，也就是css样式。

window.getComputedStyle(ele, null);

window.getComputedStyle(div, “after”); 获取伪元素

计算样式只读

返回的计算样式的值都是绝对值，没有相对单位。

IE8以及以下的版本不兼容这个方法。（ele.currentStyle 查询样式，IE独有）

**更改样式的状态，可以先将两种状态写好，然后改变类名。**

**事件绑定**

1、在DOM元素中直接绑定

在元素行间写，onclick=””可以直接写处理，也可以写函数名，函数在JS中

2、在JS代码中绑定

句柄

div.onclick = function(){}; 只能有一个操作，多个，后面覆盖前面

3、绑定事件监听函数

用 addEventListener() 或 attachEvent() 来绑定事件监听函数。

可以给一个对象绑定多个处理函数，且按顺序执行

addEventListener(type, fn, false)

IE9以下不兼容

用一个函数只能绑定一次

attachEvent(‘on’+type, fn)

IE独有 this指向window

用一个函数能绑定多次

**解除事件绑定**

div.onclick = function(){}; div.onclick = null;

addEventListener(type, fn, false) removeEventListener(type, fn, false)

attachEvent(‘on’+type, fn) detachEvent(‘on’+type, fn)

注意

若绑定匿名函数，则无法解除

**事件处理模型—冒泡、捕获**

事件冒泡

代码结构上（非视觉上）嵌套关系的元素，会存在事件冒泡的功能，即同一事件，自子元素冒泡向父元素。（自底向上）

addEventListener(type, fn, false) 冒泡

focus聚焦 blur change submit reset select 等事件没有冒泡

事件捕获

结构上（非视觉上）嵌套关系的元素，会存在事件捕获的功能，即同一事件，自父元素捕获至子元素（事件源元素）。（自顶向下）

addEventListener(type, fn, true) 捕获

IE没有捕获事件

冒泡、捕获都是指的对父级元素的处理，不包括触发事件的那个元素

元素的执行就是正常执行的

**同一个对象，定义了一个事件处理类型，两个处理函数，模型不同**

**触发顺序，先捕获，后冒泡**

**阻止事件冒泡**

W3C标准 event.stopPropagation(); 但不支持ie9以下版本

IE独有event.cancelBubble = true;

封装取消冒泡的函数 stopBubble(event)

**阻止默认事件**

默认事件 — 表单提交，a标签跳转，右键菜单等

1.return false; 以对象属性的方式注册的事件才生效

句柄 div.onclick = function(){};

2.event.preventDefault(); W3C标准，IE9以下不兼容

3.event.returnValue = false; 兼容IE

封装阻止默认事件的函数 cancelHandler(event);

封装一个方法，记住方法调用完了在外面还是要return false；

阻止a标签的跳转和跳到页面顶部的常用写法：

<a href="javascript:void(false)">点我没反应</a>

**事件对象**

event是事件对象。即：用来获取事件的详细信息：鼠标位置（clientX、clientY）、键盘按键，事件源对象 等

window.event 用于IE

**事件源对象**

在IE下，event对象有srcElement属性

Firefox下，even对象有target属性

Chrome event. srcElement 和 event. target 都有

兼容写法

var event = e || window.event;

var target = event.srcElement || event.target;

**事件委托**

将本来需要 A 处理的事情，委托给 B 来处理。

利用事件冒泡，和事件源对象进行事件处理

优点：

1、性能 不需要循环所有的元素一个个绑定事件

2、灵活 当有新的子元素时不需要重新绑定事件

**事件分类**

**1、鼠标事件**

click、dblclick、mouseover、mouseout、mousedown、mouseup、mousemove、mouseenter、mouseleave、contextmeuse

click = mousedown+mouseup 顺序mousedown mouseup click

DOM3标准规定：click事件只能监听左键

mouseup、mousedown可以区分鼠标左右键 利用 button的0、1、2

**2、键盘事件**

keydown: 键盘按下时触发

keypress:键盘按下并松开的瞬间触发

keyup: 键盘抬起时触发

【注意事项】

①执行顺序: keydown -> keypress -> keyup

②当长按时: 会循环执行 keydown -> keypress

③有 keydown 不一定有 keyup,事件触发过程中，鼠标将光标移走,将导致没有 keyup

④keypress 只能捕获键盘上的字符、数字、符号键，不能捕获各种功能键，而keydown 和 keyup 可以

⑤keypress 区分大小写，keydown 和 keyup 都不支持

**[确定触发的按键]**

①在触发的函数中,传入一个参数e,表示键盘事件

②使用 e.keyCode,取到按键的ASCII值,进而确定触发按键

③所有浏览器的兼容写法(一般不需要)

var evn=e || event;

var code=evn.keyCode || evn.which || evn.charCod；

**3、文本操作事件**

input 事件

若文本框绑定了该事件的功能, 但凡文本框内的内容出现变化, 都会触发该事件所对应的函数

change 事件

若文本框对该事件绑定了功能函数, 则当鼠标从文本框失去聚焦时, 即完成输入后点击其它地方时，事件会先对前后输入的值进行比较, 然后若文本前后发生变化, 则触发该事件绑定的功能函数.

focus 事件

鼠标聚焦事件, 当鼠标点击或聚焦于输入框内时, 触发该事件所绑定的功能函数

blur 事件

鼠标失去焦点事件, 当鼠标离开输入框, 点击其它地方时, 触发该事件所绑定的功能函数

**4、HTML事件**

scroll 事件

当滚动条一滚动, 就触发事件

load 事件

整个页面完全被加载后, 这个事件才执行, 效率很低

**JSON**

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。是一种传输数据的格式（以对象为样本，本质上就是对象，但用途有区别，对象就是本地用的，json是用来传输的）

JSON.parse (string) string --> JSON

JSON.stringify (obj) JSON -->string

**domTree + cssTree = randerTree 渲染**

**reflow 重排**

dom节点的删除、添加

dom节点的宽高变化、位置变化，display none ->block

offsetWidth、offsetLeft

**repaint 重绘** 页面的一部分

**异步加载JS**

Javascript 异步加载的三种方案

1.defer 异步加载，但要等到dom文档全部解析完才会被执行。只有IE能用。

<script type="text/javascript" src="tools.js" defer="defer"></script>

js代码可以写在标签中或者外部脚本

2.async 异步加载，加载完就执行

<script src="demo.js" aysnc="aysnc"></script>

async只能加载外部脚本，不能把js写在script 标签里。

1和2 执行时也不阻塞页面

3.创建script，插入到DOM中，加载完毕后callBack

**JS加载时间线**

在js加载开始的时候，浏览器会记录js执行的这段过程。

1、创建Document对象，开始解析web页面，解析HTML元素和他们的文本内容后添加Element对象和Text节点到文档中。

这个阶段document.readyState = "loading"。

2、遇到link外部css，创建线程加载，并继续解析文档。

3、遇到script外部js，并且没有设置async，defer，浏览器加载，并阻塞，等待js加载完成并执行该脚本，然后继续解析文档

4、遇到script外部js，并且设置有async，defer 浏览器创建线程加载，并继续解析文档，对于async属性的脚本，脚本加载完成后立即执行

（异步禁止使用docuemnt.write（）会消除文档流）

5、遇到img标签等，先正常解析dom结构，然后浏览器异步加载src，并继续解析文档

6、当文档解析完成，document.readyState = "interactive"；

7、文档解析完成后，所有设置有defer的脚本会按照顺序执行。

8、当文档解析完成之后，document对象触发DOMContentLoaded事件

这也标志着程序执行从同步脚本执行阶段，转化为事件驱动阶段

9、当所有saync的脚本加载完成并执行后，img等加载完成后，

document.readyState = "complete"

window对象触发load事件

10、从此，页面以异步响应方式处理用户输入，网络事件等。

**正则表达式**

手机屏幕截图

描述已自动生成1、RegExp对象相关方法

手机屏幕截图

描述已自动生成2、String对象相关方法