Web API

API的概念

API（Application Programming Interface,应用程序编程接口）是一些预先定义的函数，目的是提供应用程序与开发人员基于某软件或硬件得以访问一组例程的能力，而又无需访问源码，或理解内部工作机制的细节。

- 任何开发语言都有自己的API
- API的特征输入和输出(I/O)
- API的使用方法(console.log())

Web API的概念

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)

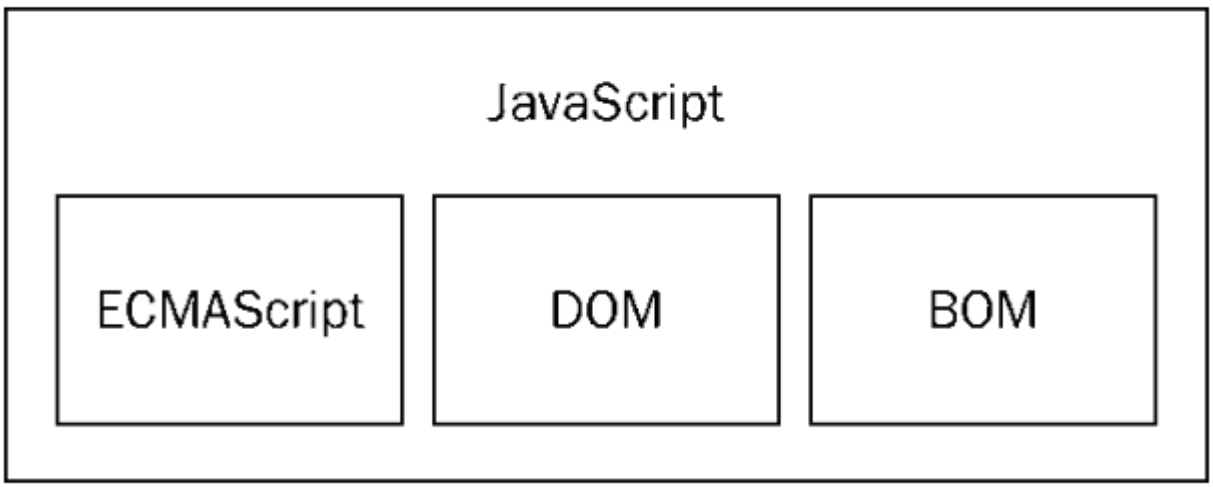
此处的Web API特指浏览器提供的API(一组方法)，Web API在后面的课程中有其它含义

掌握常见的浏览器提供的API的调用方式

利用javascript调用DOM 或者 BOM 提供的API方法 来实现通过js 操作页面上的标签 文档 或者浏览器的功能

MDN-Web API

JavaScript的组成



ECMAScript - JavaScript的核心

定义了javascript的语法规范

JavaScript的核心，描述了语言的基本语法和数据类型，ECMAScript是一套标准，定义了一种语言的标准与具体实现无关

BOM - 浏览器对象模型

一套操作浏览器功能的API

通过BOM可以操作浏览器窗口，比如：弹出框、控制浏览器跳转、获取分辨率等

DOM - 文档对象模型

一套操作页面元素的API

DOM可以把HTML看做是文档树，通过DOM提供的API可以对树上的节点进行操作

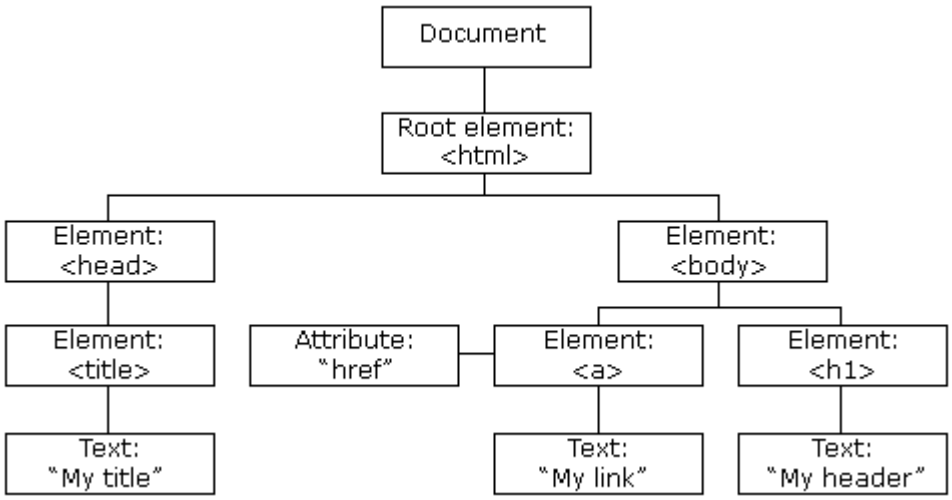
JavaScript DOM开发

> 当网页被加载时，浏览器会创建页面的文档对象模型（Document Object Model），DOM 是 W3C（万维网联盟）的标准。DOM 定义了访问 HTML 和 XML 文档的标准>。
>
> 我们经常使用JavaScript来操纵DOM，不过DOM其实的语言无关的。它并不是JavaScript的一部分。

官方地址:

HTML DOM 模型被构造为对象的树：DOM树

```
<!DOCTYPE html>
<html>
<head>
  <title>My title</title>
</head>
<body>
  <a href="">My link</a>
  <h1>My header</h1>
</body>
</html>
```



DOM基础名词

- 文档：一个网页可以称为文档
- 节点：网页中的所有内容都是节点（标签、属性、文本、注释等）
- 元素：网页中的标签
- 属性：标签的属性

DOM API分类

> DOM的学习主要围绕操作 标签节点与用户事件交互两个方面，我们先看一下所有需要学习的DOM API分类

####

DOM的属性 节点名称 节点类型 节点
对元素对象的 增 删 改 查
对元素属性的 增 删 改 查
对元素位置的获取
键盘事件与鼠标事件 事件监听 事件方法 事件对象 [点击,滚动,移入,移出,输入,键入]

Node(节点)基础分类

> DOM1级定义了一个Node接口，该接口由DOM中所有节点类型实现。这个Node接口在JS中是作为Node类型实现的。在IE9以下版本无法访问到这个类型，JS中所有节点都继承自Node类型，都共享着相同的基本属性和方法。
> Node有一个属性nodeType表示Node的类型，它是一个整数，其数值分别表示相应的Node类型，具体如下：

```
Node.ELEMENT_NODE:1 //标签 *
Node.ATTRIBUTE_NODE:2 //属性 *
Node.TEXT_NODE:3 //文本 *
Node.CDATA_SECTION_NODE:4 //子节点一定为TextNode
Node.ENTITY_REFERENCE_NODE:5
Node.ENTITY_NODE:6
Node.PROCESSING_INSTRUCTION_NODE:7 //命令节点
Node.COMMENT_NODE:8  //注释
Node.DOCUMENT_NODE:9  //最外层的Root element,包括所有其它节点 *
Node.DOCUMENT_TYPE_NODE:10  // DTD，<!DOCTYPE.....>
Node.DOCUMENT_FRAGMENT_NODE:11 //文档片段节点
Node.NOTATION_NODE:12 //DTD中的Nation定义
```

Element类型

Element提供了对元素标签名，子节点和特性的访问，我们常用HTML元素比如div，span，a等标签就是element中的一种。Element有下面几条特性：

（1）nodeType为1
（2）nodeName为元素标签名，tagName也是返回标签名
（3）nodeValue为null
（4）parentNode可能是Document或Element
（5）子节点可能是Element，Text，Comment，Processing_Instruction，CDATASection或EntityReference

Text类型

Text表示文本节点，它包含的是纯文本内容，不能包含html代码，但可以包含转义后的html代码。Text有下面的特性：

（1）nodeType为3
（2）nodeName为#text
（3）nodeValue为文本内容
（4）parentNode是一个Element
（5）没有子节点

Attr类型

Attr类型表示元素的特性，相当于元素的attributes属性中的节点，它有下面的特性：

（1）nodeType值为2
（2）nodeName是特性的名称
（3）nodeValue是属性的值
（4）parentNode为null

Comment类型

Comment表示HTML文档中的注释，它有下面的几种特征：

（1）nodeType为8
（2）nodeName为#comment
（3）nodeValue为注释的内容
（4）parentNode可能是Document或Element
（5）没有子节点

Document

Document表示文档，在浏览器中，document对象是HTMLDocument的一个实例，表示整个页面，它同时也是window对象的一个属性。Document有下面的特性：

（1）nodeType为9
（2）nodeName为#document
（3）nodeValue为null
（4）parentNode为null
（5）子节点可能是一个DocumentType或Element

DocumentType

DocumentType表示文档的DTD声明,用于确定文档版本,确定对应的API集与属性解析规则：

（1）nodeType为10
（2）nodeName为#document-fragment
（3）nodeValue为null
（4）parentNode为null

DocumentFragment类型

DocumentFragment是所有节点中唯一一个没有对应标记的类型，它表示一种轻量级的文档，可能当作一个临时的仓库用来保存可能会添加到文档中的节点。DocumentFragment有下面的特性：

（1）nodeType为11
（2）nodeName为#document-fragment
（3）nodeValue为null
（4）parentNode为null

DOM API 节点对象选择器

> 通过以下方法可以获取DOM节点对象

```
var oHeader = document.getElementById('header');
//获取ID名称为 header 的标签 返回对应的节点对象(单个) 如果文档内有多个 ID名称为 header 的标签 只获取第一个

var aP = document.getElementsByTagName('p');
//获取文档内的所有 p 标签 返回一个 DOM集合(NodeList) 类数组对象 哪怕没有找到只有一个也返回类数组集合

var aDes = document.getElementsByClassName('des'); //IE9
//获取文档内的所有类名为 des 的标签 返回一个 DOM集合(NodeList) 类数组对象 哪怕没有找到只有一个也返回类数组集合

var oHeader = document.querySelector('#header');
//querySelector是H5 新增的DOM API 参数使用合法的css选择器即可  返回复合条件的第一个元素 (唯一)
```

```
var aDes = document.querySelectorAll('.des');
//querySelectorAll是H5 新增的DOM API 参数使用合法的css选择器即可  返回复合条件的节点集合 类数组对象(非唯一)

var body = document.body; //直接获取body节点对象
```

DOM 节点对象属性

> DOM节点对象拥有一系列属性 用于存储该节点的状态 信息

```
element.title //设置或返回元素的title属性
element.textContent //设置或返回一个节点和它的文本内容
element.innerText //设置或返回一个节点和它的文本内容
element.tagName //作为一个字符串返回某个元素的标记名（大写）
element.className //获取标签的class属性值

nodelist.length //返回节点列表的节点数目。
nodelist.item(idx) //返回某个元素基于文档树的索引 同 nodelist[idx]
```

DOM API 获取标签属性

> 通过以下方法可以获取DOM元素的属性

```
var oHeader = document.getElementById('header');
var aImg = document.getElementsByTagName('img');

console.log(oHeader.attributes); //节点属性对象 拥有长度属性

console.log(oHeader.id); //指定属性获取

console.log(aImg[0].src);//指定属性获取

console.log(aImg.getAttribute('src')); //通过 getAttribute方法获取实际 属性值

console.log(aImg.hasAttribute('src')); // 判断节点对象是否含有 某个 属性
```

DOM API 获取样式

> DOM样式获取分为获取 行内style样式 和 实际计算后样式两种

```
//行内样式
console.log(oImg.style); //CSSOM对象
console.log(aImg.style.outline); //标签行内样式 style属性中存在的样式的值

//实际样式
console.log(window.getComputedStyle(oImg, null)['border']); //主流浏览器
console.log(oImg.currentStyle['border']); //老版本IE

//兼容函数写法
function getStyle(obj, attr) {
   return obj.currentStyle ? obj.currentStyle[attr] : getComputedStyle(obj, false)[attr];
}
```

DOM API 获取节点内容

> 获取节点内容主要掌握 获取节点文本内容 获取实际内容两种

```
console.log(oHeader.innerHTML); //获取标签内的实际内容(包括标签)

 console.log(oHeader.innerText);  //设置标签中间的文本内容，应该使用innerText属性,谷歌火狐支持，ie8支持
 console.log(oHeader.textContent); //设置标签中间的文本内容，应该使用textContent属性,谷歌火狐支持，ie8不支持


//textContent 与 innerText兼容处理
function getInnerText(element) {
   if(typeof element.textContent=="undefined"){
      return element.innerText;
   }else{
      return element.textContent;
   }
}
```

DOM API 获取节点相关节点

> 我们可以通过方法获取节点的相关联节点 后代 父亲 兄弟等

```
console.log(oHeader.children); //获取子元素 只有标签
console.log(oHeader.childNodes); //获取子节点 包含文本节点与标签节点

console.log(oHeader.firstChild); //获取第一个子节点(包含文本)
```

```
console.log(oHeader.firstElementChild); //获取第一个子标签节点


console.log(oHeader.lastChild); //获取最后一个子节点(包含文本)
console.log(oHeader.lastElementChild); //获取最后一个子标签节点

console.log(aP[0].parentElement); //父元素
console.log(aP[0].parentNode); //父节点

console.log(aP[1].nextElementSibling); //下一个兄弟标签节点
console.log(aP[1].nextSibling); //下一个兄弟节点(计算文本节点)


console.log(aP[2].previousElementSibling); //上一个兄弟标签节点
console.log(aP[2].previousSibling); //上一个兄弟节点(计算文本节点)
```

DOM API 创建与添加节点标签

> 通过相关方法可以创建节点 添加节点到HTML文档中

```
var cP = document.createElement("p"); //创建标签节点

var content = document.createTextNode("你好"); //创建文本节点

cP.appendChild(content); //添加content到cP节点中
document.body.appendChild(cP); //添加cP到body标签中进行渲染
document.body.append(cP); //append 是H5 WEB API 新增方法

----------------------------------------

//通过文本标签方式添加节点
var htmlTxt = '<p>哈哈哈</p>';
document.body.innerHTML += htmlTxt;

//HTML输出流 直接覆盖body中的内容
document.write('<p>123</p>');

----------------------------------------
//在父节点ELE里面的节点A前面添加 新的节点B

ELE.insertBefore(B,A);
```

DOM API 节点替换拷贝

> 通过相关方法可以实现 节点拷贝 节点

```
//复制拷贝

var oWrap = document.getElementById('wrap');
//cloneNode方法会对调用它的节点对象进行复制 传参true代表包括该节点的后代节点 不传参数表示只复制该节点本身
var cloneWrap = oWrap.cloneNode(true);
document.body.appendChild(cloneWrap);



----------------------------------------------------------
//替换
var oWrap = document.querySelector('#wrap');
var oDes = oWrap.querySelector('#wrap .des');
var newDes = document.createElement('p');
newDes.innerHTML = '我会替换掉你';
oWrap.replaceChild(newDes,oDes);

// 用newDes替换oWrap内部的oDes
```

DOM API 节点删除

> 通过如下方法可以实现DOM节点的删除

```
element.removeChild(element.children[1]); //element删除了element内的下标为1的子元素
element.remove(); //H5 DOM API 自己删自己
```

DOM API 节点检查

> 通过如下方法可以对Node节点对象进行一系列检测

```
document.hasFocus(); //返回布尔值，检测文档或元素是否获取焦点
element.hasChildNodes(); //返回布尔值,检测节点对象是否包含任何子节点
element.isEqualNode(element2); //返回布尔值,判断element与element2是否是同一个节点
element.hasAttributes(); //返回布尔值,判断当前节点对象是否包含属性
element.hasAttribute(property); //返回布尔值, 判断该节点是否拥有指定的 property 属性
```

基础事件

> 用户与文档交互基础事件分为鼠标事件与键盘事件

例如:

- 当用户点击鼠标时
- 当网页加载后
- 当图像加载后
- 当鼠标移至元素上时
- 当输入字段被改变时
- 当 HTML 表单被提交时
- 当用户敲击按键时

事件句柄

属性	此事件发生在何时...
onabort	图像的加载被中断。
onblur	元素失去焦点。
onchange	域的内容被改变。
onclick	当用户点击某个对象时调用的事件句柄。
ondblclick	当用户双击某个对象时调用的事件句柄。
onerror	在加载文档或图像时发生错误。
onfocus	元素获得焦点。
onkeydown	某个键盘按键被按下。
onkeypress	某个键盘按键被按下并松开。
onkeyup	某个键盘按键被松开。
oninput	input输入框接收到输入内容时触发 H5 API
onload	一张页面或一幅图像完成加载。
onmousedown	鼠标按钮被按下。
onmousemove	鼠标被移动。
onmouseover	鼠标移到某元素之上。
onmouseout	鼠标从某元素移开。
onmouseenter	鼠标移到某元素之上。(不支持冒泡)
onmouseleave	鼠标从某元素移开。(不支持冒泡)
onmouseup	鼠标按键被松开。
onreset	重置按钮被点击。
onresize	窗口或框架被重新调整大小。
onselect	文本被选中。
onsubmit	确认按钮被点击。
onunload	用户退出页面。

```
var oBtn = document.querySelector('#btn');

oBtn.onclick = function(e){
   console.log(e)
  this.value = '不要点我';
}

//事件回调函数会回调形参 e Event对象
```

事件：触发-响应机制

Event接口表示在DOM中发生的任何事件，一些是用户生成的（例如鼠标或键盘事件），而其他由API生成。

事件三要素

- 事件源:触发(被)事件的元素
- 事件类型:事件的触发方式(例如鼠标点击或键盘点击)
- 事件处理程序:事件触发后要执行的代码(函数形式)

事件阶段

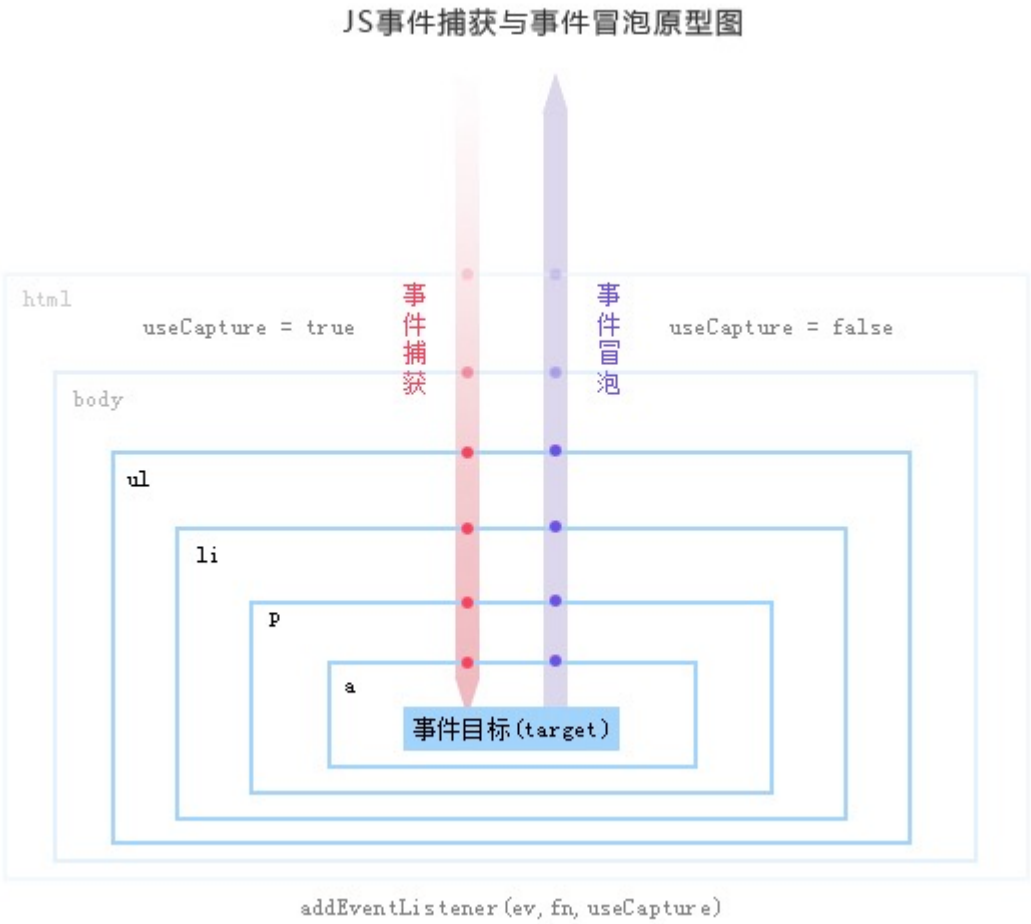
事件有三个阶段:

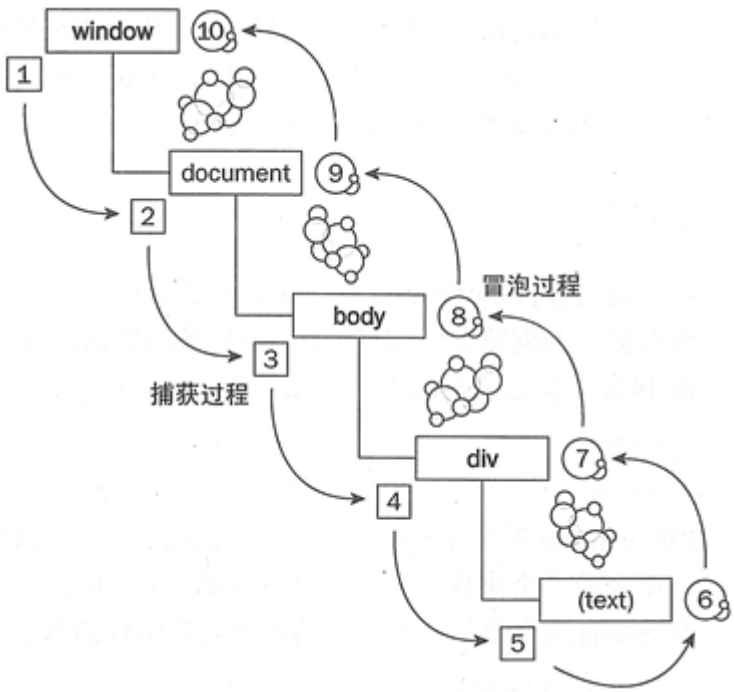
event.eventPhase属性可以查看事件触发时所处的阶段 :

1. 捕获阶段
2. 当前目标阶段
3. 冒泡阶段

> **1、事件捕获**
> 捕获型事件(event capturing)：事件从最不精确的对象(document 对象)开始触发，然后到最精确(也可以在窗口级别捕获事件，不过必须由开发人员特别指定)
>
> **2、事件冒泡**
> 冒泡型事件：事件按照从最特定的事件目标到最不特定的事件目标(document对象)的顺序触发。

绑定的事件默认的执行时间是在冒泡阶段执行，而非在捕获阶段（重要）。这也是为什么当父类和子类都绑定了某个事件，会先调用子类绑定的事件，后调用父类的事件





事件对象的属性和方法

- event.type 获取事件类型
- clientX/clientY　所有浏览器都支持，窗口位置
- pageX/pageY　IE8以前不支持，页面位置

- event.target || event.srcElement 用于获取触发事件的元素
- event.preventDefault() 取消默认行为

event对象属性

属性	描述
altKey	返回当事件被触发时，"ALT" 是否被按下。
shiftKey	返回当事件被触发时，"SHIFT" 键是否被按下。
ctrlKey	返回当事件被触发时，"CTRL" 键是否被按下。
metaKey	返回当事件被触发时，"meta" 键是否被按下。
button	返回当事件被触发时，哪个鼠标按钮被点击。
clientX	返回当事件被触发时，鼠标指针的水平坐标。
clientY	返回当事件被触发时，鼠标指针的垂直坐标。
keyCode	表示非字符按键的unicode值。
isChar	布尔值，表示当前按下的键是否表示一个字符
screenX	返回当某个事件被触发时，鼠标指针的水平坐标。
screenY	返回当某个事件被触发时，鼠标指针的垂直坐标。
pageX	事件发生时相对于页面（如viewport区域）的水平坐标。
pageY	事件发生时相对于页面（如viewport区域）的垂直坐标。
currentTarget	事件冒泡阶段所在的当前DOM元素
target	返回触发此事件的元素（事件的目标节点）。
eventPhase	返回事件传播的当前阶段。
cancelable	返回布尔值，指示事件是否可拥可取消的默认动作。
type	返回当前 Event 对象表示的事件的名称。
timeStamp	返回事件生成的日期和时间。

事件行为

阻止默认事件行为

> 很多标签拥有默认的事件行为 比如a标签 点击会执行跳转页面行为 , 我们可以通过代码阻止这些行为

```
var oLink = document.querySelector('a');

oLink.onclick = function(e){
  e.preventDefault();
    //preventDefault它是事件对象(Event)的一个方法，作用是取消一个目标元素的默认行为
}

oLink.onclick = function(e){
  //代码
    return false; //回调函数最后返回false可以阻止默认行为
}
```

阻止事件传递(阻止冒泡)

> 所有的事件类型都会经历事件捕获但是只有部分事件会经历事件冒泡阶段,例如submit事件就不会被冒泡。

```
事件的传播是可以阻止的：
w3c规范下，使用stopPropagation（）方法
在IE版本下设置eve.cancelBubble = true； 废弃
在捕获的过程中stopPropagation（）；后，后面的冒泡过程就不会发生了。

var oLink = document.querySelector('a');

oLink.onclick = function(e){
  e.stopPropagation();

}
```

事件解绑

> 事件绑定之后 如果需要解绑 可以销毁

```
var oBtn = document.querySelector('#btn');
var oClean = document.querySelector('#clean');

oBtn.onclick = function(){
    console.log('btn成功绑定点击事件');
}
oClean.onclick = function(){
    oBtn.onclick = null;
    console.log('btn已解绑点击事件');
}
```

事件委托

> 当需要对多个子元素进行循环事件绑定的时候，可以将事件委托与他们的共同父级，通过event对象属性来进行筛选和操作，节省开销。

```
var oUl = document.querySelector('list');

//监听在oUl上发生的点击事件 利用事件的传递性 获取e.target判断触发事件的实际DOM是否为li
oUl.onclick = function(e){
   if(e.target.toLowerCase() === 'li'){
     e.target.style.backgroundColor = '#368';
   }
}
```

事件分派

> 当一个容器有多个元素需要绑定同一个事件, 而点击不同元素所需要执行的操作不同时可以进行分派映射

```
 var oBox = document.querySelector('#box');

var eventMap = {
   'stretch': function (ele) {
      ele.style.width = '400px';
      ele.style.height = '400px';
   },
   'discolor': function (ele) {
      ele.style.backgroundColor = '#368';
   },
   'rotate': function (ele) {
      ele.style.transform = 'rotate(10deg)';
   }
}

document.onclick = function (e) {
   if (eventMap[e.target.id.toLowerCase()]) {
      eventMap[e.target.id.toLowerCase()](oBox);
   }
}
```

监听器

> DOM 事件监听器 addEventListener() 方法 与 attachEvent()
>
> addEventListener() 允许您将事件监听器添加到任何 HTML DOM 对象上，比如 HTML 元素、HTML 对象、window 对象或其他支持事件的对象，比如 xmlHttpRequest 对象。

事件监听

```
//监听窗口大小变化
window.addEventListener("resize", function(){
   console.log(window.innerWidth,window.innerHeight); // document.documentElement.width
},false);

-----------------------------------------------

oWrap.addEventListener('click',touch,false);
oWrap.attachEvent("onclick",touch);

function touch(e){
   //do somthing
}

注意:
1. addEventListener用于标准浏览器 attachEvent用于老版本IE浏览器
2. addEventListener参数依次为 事件名称(不加on) 事件触发函数 事件触发阶段(true:捕获,false:冒泡)
3. addEventListener 回调函数内部this指向 绑定对象
```

解除监听

```
oWrap.removeEventListener("click",touch,false);

oWrap.detachEvent("onclick",touch);


注意:用什么方式绑定事件,就应该用对应的方式解绑事件
1.
对象.on事件名字=事件处理函数--->绑定事件
对象.on事件名字=null;
2.
对象.addEventListener("没有on的事件类型",命名函数,false);---绑定事件
对象.removeEventListener("没有on的事件类型",函数名字,false);
3.
对象.attachEvent("on事件类型",命名函数);---绑定事件
对象.detachEvent("on事件类型",函数名字);
```

事件监听与on绑定区别

> on事件会被后面的on的事件覆盖, addEventListener 则不会覆盖;
>
> addEventListener可以指定事件回调触发时机 (捕获 or 冒泡) on事件只有 冒泡时刻触发
>
> addEventListener本质是**函数定义与具体调用事件的解耦，实现了同一事件可以调用多个函数，同一函数可以被多个事件调用，推荐使用*。

兼容写法

```
//为任意一个元素绑定事件:元素,事件类型,事件处理函数
function addEventListener(element,type,fn) {
   if(element.addEventListener){
      //支持
      element.addEventListener(type,fn,false);
   }else if(element.attachEvent){
      element.attachEvent("on"+type,fn);
   }else{
      element["on"+type]=fn;
   }
}


//为任意的一个元素解绑某个事件:元素,事件类型,事件处理函数
function removeEventListener(element,type,fn) {
   if(element.removeEventListener){
      element.removeEventListener(type,fn,false);
   }else if(element.detachEvent){
      element.detachEvent("on"+type,fn);
   }else{
      element["on"+type]=null;
   }
}
```

方法区别

```
1.方法名不一样
2.参数个数不一样addEventListener三个参数,attachEvent两个参数
3.addEventListener 谷歌,火狐,IE11支持,IE8不支持
attachEvent 谷歌火狐不支持,IE11不支持,IE8支持
4.this不同,addEventListener 中的this是当前绑定事件的对象
attachEvent中的this是window
5.addEventListener中事件的类型(事件的名字)没有on
attachEvent中的事件的类型(事件的名字)有on
```

匿名解绑监听

```
var btn = document.getElementById('btn');
var cancel = document.getElementById('cancel');
var listenBySL = function(element, type, handler, capture){
  capture = capture || false;
   if(element.addEventListener){
      // W3C内核
```

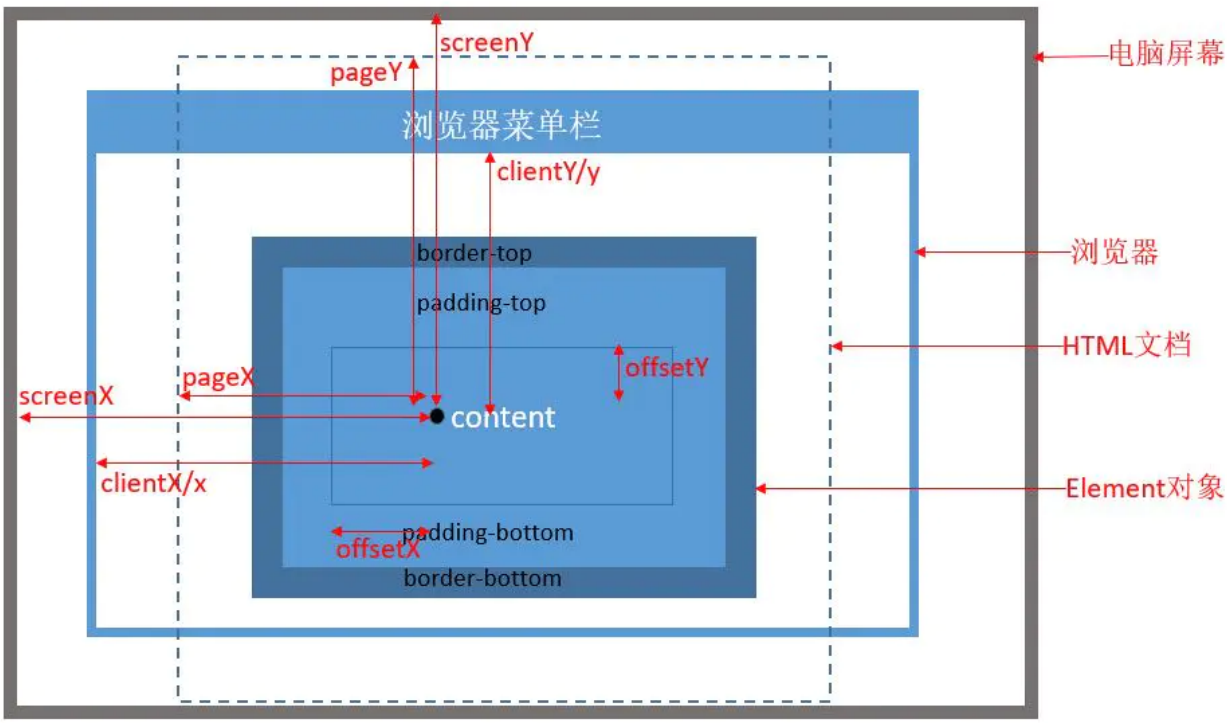
```
        element.addEventListener(type, handler, capture);
    }else{
        // IE内核
        element.attachEvent('on'+type, handler, capture);
    }

    return {
        "remove":function(){
            if(element.removeEventListener){
                // W3C内核
                element.removeEventListener(type, handler, capture);
            }else{
                // IE内核
                element.detachEvent('on'+type, handler, capture);
            }
        }
    }
}
// 添加监听
var addAlert = listenBySL(btn,'click',function(){
    alert(123);
});
listenBySL(cancel,'click',function(){
    // 移除监听
    addAlert.remove();
    alert('移除成功');
});
```

鼠标Event 位置

> 鼠标event对象给我们展示了 client offset page screen系列的X Y坐标 可以用于在不同需求下获取鼠标的相对位置

属性	说明
clientX	以浏览器左上顶角为原点，定位 x 轴坐标
clientY	以浏览器左上顶角为原点，定位 y 轴坐标
offsetX	以当前事件的目标对象content左上角为原点，定位x轴坐标
offsetY	以当前事件的目标对象content左上角为原点，定位y轴坐标
pageX	以Document 对象（即文本窗口）左上角为原点，定位x轴坐标
pageY	以Document 对象（即文本窗口）左上角为原点，定位 y 轴坐标
screenX	计算机屏幕左上角为原点，定位x轴坐标
screenY	计算机屏幕左上角为原点，定位 y 轴坐标
layerX	最近的绝对定位的父元素（如果没有，则为Document对象）左上角为原点，定位x轴坐标
layerY	最近的绝对定位的父元素（如果没有，则为Document对象）左上角为原点，定位 y 轴坐标

图例



兼容

	chrome	firefox	ie8 -	ie9	ie10 +
offsetX	√	×	√	√	√
clientX	√	√	√	√	√
pageX	√	√	×	√	√
screenX	√	√	√	√	√
layerX	√	√	×	√	√
x	√	×	√	√	√

定时器

setTimeout()和clearTimeout()

在指定的毫秒数到达之后执行指定的函数，只执行一次

```
// 创建一个定时器，1000毫秒后执行，返回定时器的标示
var timerId = setTimeout(function () {
  console.log('Hello World');
}, 1000);

// 取消定时器的执行
clearTimeout(timerId);
```

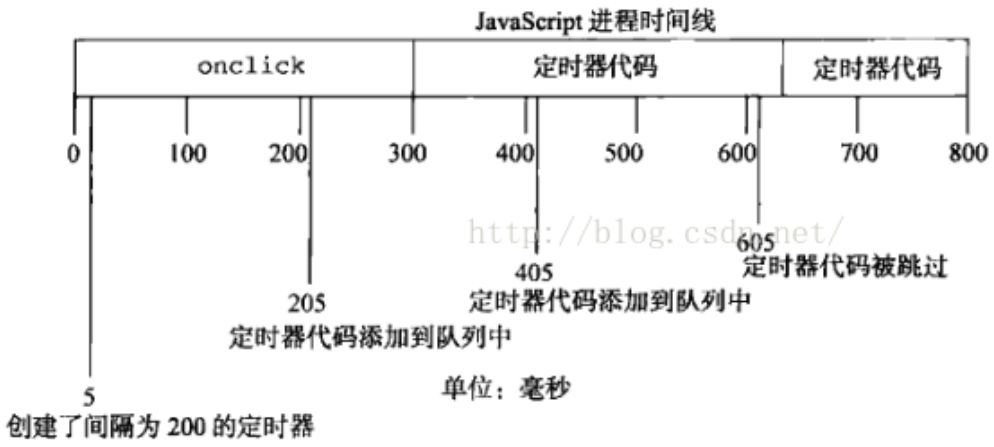
setInterval()和clearInterval()

定时调用的函数，可以按照给定的时间(单位毫秒)周期调用函数

```
// 创建一个定时器，每隔1秒调用一次
var timerId = setInterval(function () {
  var date = new Date();
  console.log(date.toLocaleTimeString());
}, 1000);

// 取消定时器的执行
clearInterval(timerId);
```

注意事项

1. HTML5规范规定了最小延迟时间不得小于4ms，即如果x小于4，会被当做4来处理
2. 由于javascript单线程 执行队列需要进行插入调整 所以setInterval会出现下述问题
   （1）某些间隔会被跳过了
   （2）多个定时器的代码执行间隔可能会比预期的要小。
3. 定时器调用传入函数名称+() 会导致回调函数直接执行
4. 通过setTimeout递归自调可以替代setInterval
5. 当前页面处于hide(不可见 离开)状态时 定时器会休眠 但是队列会持续添加 会导致失序



递归setTimeout实现有序的定时序列

```
//参数： 毫秒 需要执行的方法
function setInter(s,fn){
   function timeOut(s,fn){
     setTimeout(function(){
       fn();
       timeOut(s,fn);
     },s)
  }
  timeOut(s,fn);
}
```

视窗位置与尺寸

获取视口宽高

下面方法是包括滚动条的宽高，不支持 IE8

```
window.innerWidth
window.innerHeight
```

> `width + padding + border + 滚动条` 另外 outerWidth 浏览器兼容差，可获取包括工具栏的宽高

页面滚动位置

返回整个页面的滚动的位置，pageYOffset/pageXOffset 与 scrollY/scrollX 返回的值一致，前者是后者的别名，建议使用前者，不支持 IE8

```
window.pageYOffset || document.documentElement.scrollTop || document.body.scrollTop
window.pageXOffset || document.documentElement.scrollLeft || document.body.scrollLeft
```

窗口在显示器的位置

标准浏览器使用的是 screenX/screenY，IE 中使用的是 screenLeft/screenTop

```
window.screenLeft || window.screenX
window.screenTop || window.screenY
```

###

元素占用的空间尺寸和位置

getBoundingClientRect

使用方法 `getBoundingClientRect()` 返回的值见下图：

> IE 只返回 top right bottom left 四个值，如果需要 width height 则需要计算：

```
function getBoundingClientRect(elem) {
   var rect = elem.getBoundingClientRect()
   return {
      top: rect.top,
      right: rect.right,
      bottom: rect.bottom,
      left: rect.left,
      width: rect.width || rect.right - rect.left,
      height: rect.height || rect.bottom - rect.top
   }
}
```

clientWidth/clientHeight

返回元素不含滚动条的尺寸，不包括边框

```
document.documentElement.clientWidth || document.body.clientWidth
document.documentElement.clientHeight || document.body.clientHeight
```

- 如果是 document.documentElement，那么返回的是不包含滚动条的视口尺寸
- 如果是 document.body，并且是在混杂模式下，那么返回的是不包含滚动条的视口尺寸

clientLeft/clientTop

返回的是计算后的 CSS 样式的 border-left-width/border-top-width 的值，就是边框的宽度

offsetWidth/offsetHeight

同样可以使用 offsetWidth/offsetHeight 来获取元素包括滚动条和边框的尺寸，这个方法返回元素本身的宽高 + padding + border + 滚动条

offsetLeft/offsetTop

相对于最近的祖先定位元素（CSS position 属性被设置为 relative、absolute 或 fixed 的元素）的左右偏移值

offsetLeft/offsetTop 返回元素 X Y 坐标值

计算元素的位置：

```
function getElementPosition(e) {
  var x = 0, y = 0;
  while (e != null) {
    x += e.offsetLeft;
    y += e.offsetTop;
    e = e.offsetParent; // 获取最近的祖先定位元素
  }
  return {
    x: x,
    y: y
  };
}
```

元素内容的宽高和滚动距离

scrollWidth/scrollHeight

这个方法返回元素内容区域的宽高 + padding + 溢出内容尺寸

```
document.documentElement.scrollWidth || document.body.scrollWidth
document.documentElement.scrollHeight || document.body.scrollHeight
```

- 如果元素是 document.documentElement，返回的是视口滚动区域宽度和视口宽度中较大的那个
- 如果元素是 document.body，并且是在混杂模式下，那么返回的是视口滚动区域宽度和视口宽度中较大的那个

scrollLeft/scrollTop

这个方法返回元素滚动条的位置

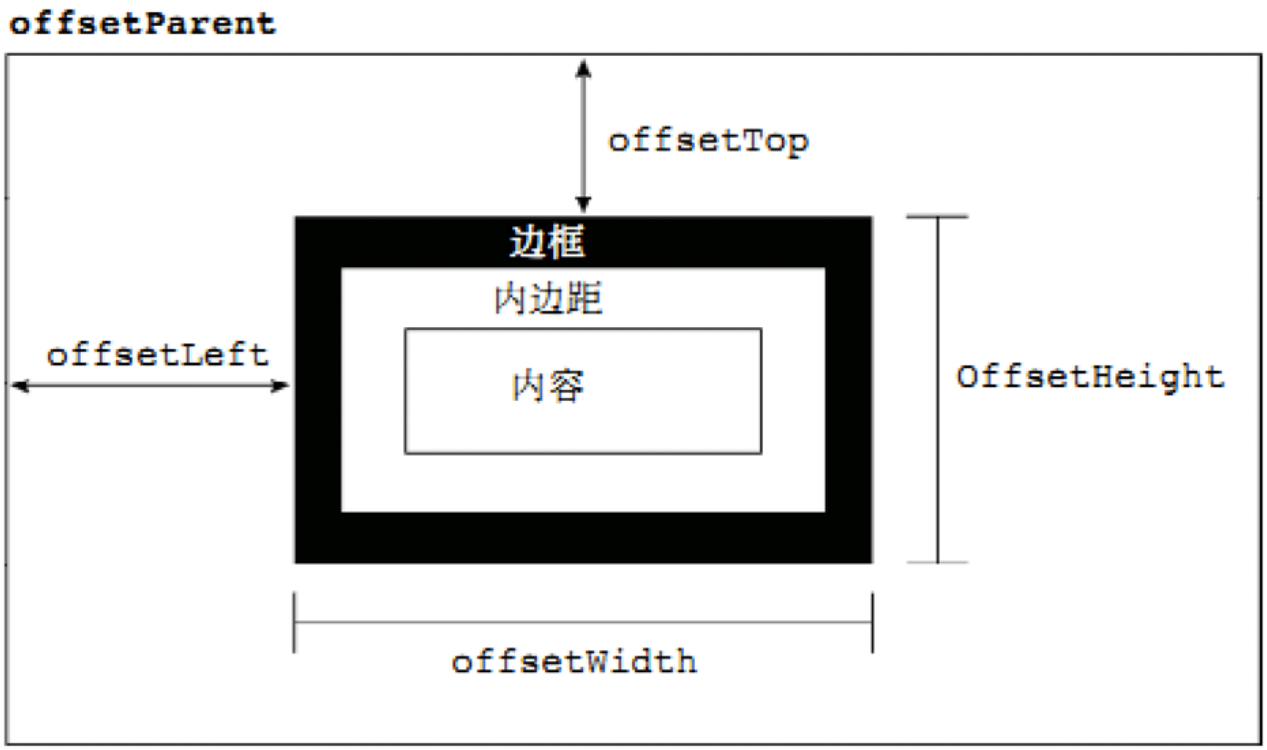
- 如果元素是根元素，那么返回 window.scrollY 的值
- 如果元素是 body，并且在混杂模式下，那么返回的是 window.scrollY 的值

> 因此可用于处理页面滚动的距离的兼容

偏移量

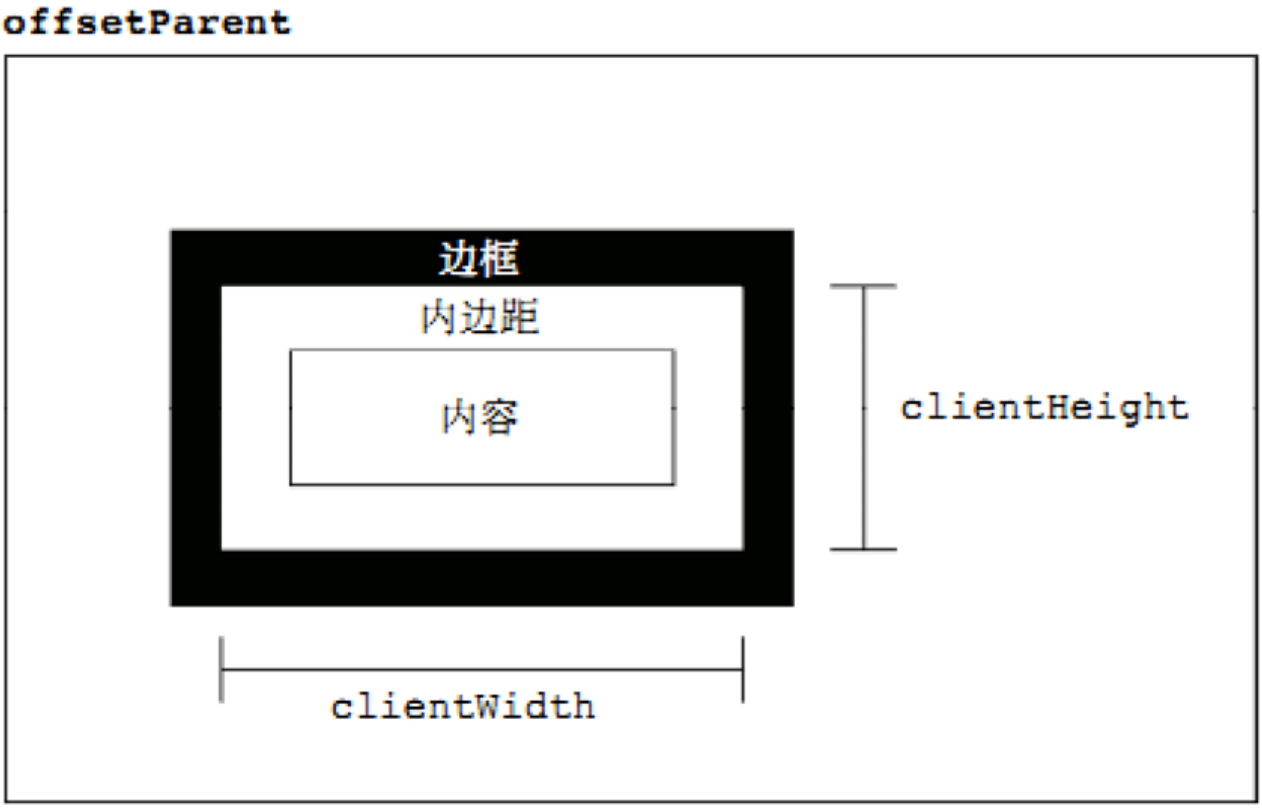
- offsetParent用于获取定位的父级元素
- offsetParent和parentNode的区别

```
var box = document.getElementById('box');
console.log(box.offsetParent);
console.log(box.offsetLeft);
console.log(box.offsetTop);
console.log(box.offsetWidth);
console.log(box.offsetHeight);
```



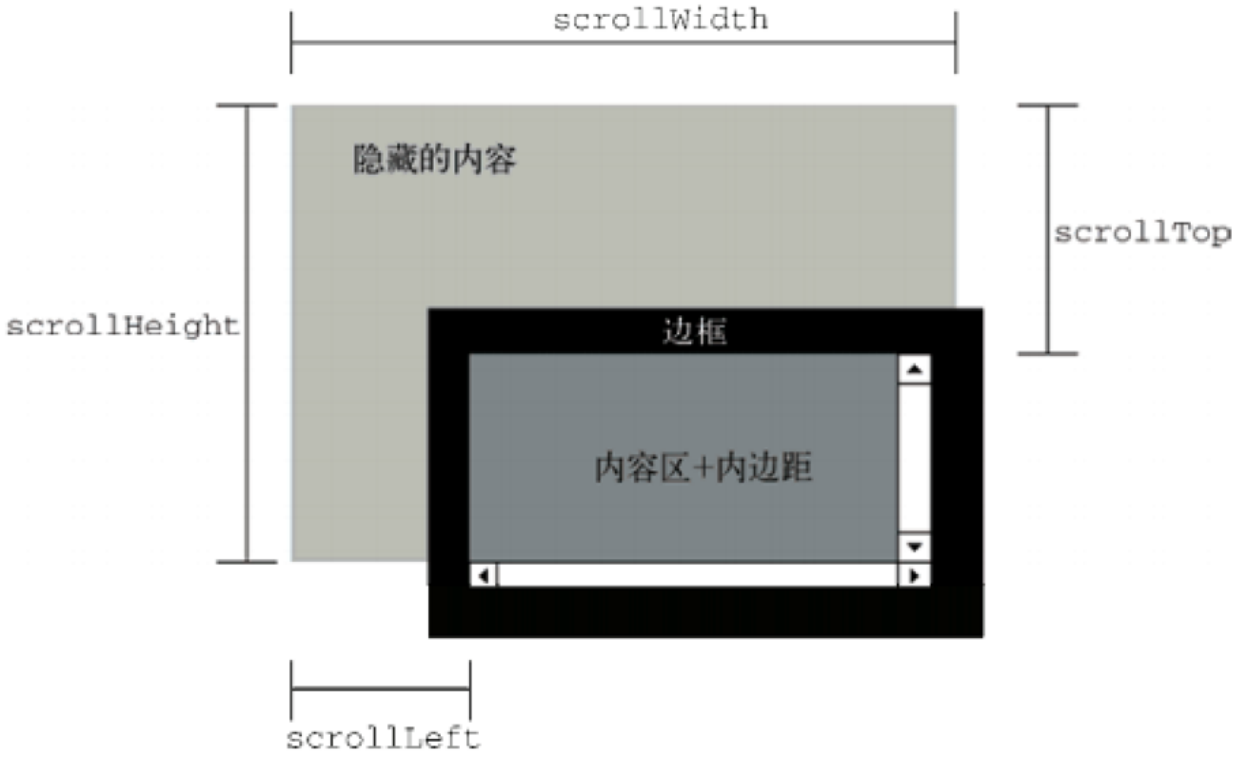
内容区域大小

```
var box = document.getElementById('box');
console.log(box.clientLeft);
console.log(box.clientTop);
console.log(box.clientWidth);
console.log(box.clientHeight);
```



滚动偏移

```
var box = document.getElementById('box');
console.log(box.scrollLeft)
console.log(box.scrollTop)
console.log(box.scrollWidth)
console.log(box.scrollHeight)
```



汇总

offset系列:获取元素的宽,高,left,top, offsetParent
offsetWidth:元素的宽,有边框
offsetHeight:元素的高,有边框
offsetLeft:元素距离左边位置的值
offsetTop:元素距离上面位置的值

scroll系列:卷曲出去的值
scrollLeft:向左卷曲出去的距离
scrollTop:向上卷曲出去的距离
scrollWidth:元素中内容的实际的宽(如果内容很少或者没有内容, 元素自身的宽),没有边框
scrollHeight:元素中内容的实际的高(如果内容很少或者没有内容,元素自身的高),没有边框

client系列:可视区域
clientWidth:可视区域的宽(没有边框),边框内部的宽度
clientHeight:可视区域的高(没有边框),边框内部的高度
clientLeft:左边边框的宽度
clientTop :上面的边框的宽度

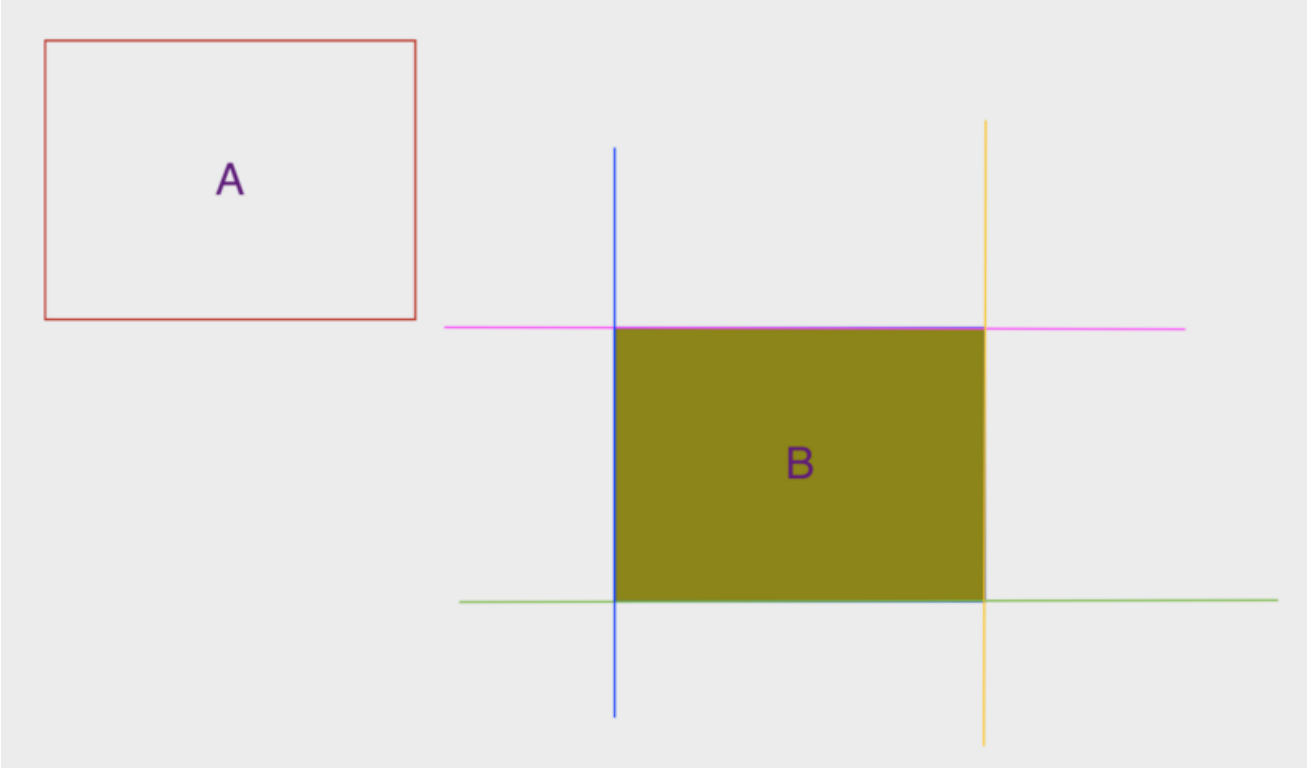
碰撞检测

> **碰撞检测**（边界检测）在前端游戏，以及涉及拖拽交互的场景应用十分广泛。
> 碰撞，顾名思义，就是两个物体碰撞在了一起，眼睛是可以直观的观察到碰撞的发生。但对于前端实现，如何让 JavaScript 代码理解两个独立的“物体”（DOM）碰撞在一起呢。这就涉及到碰撞检测（或者叫边界检测）的问题了。

两个矩形块的碰撞：

判断任意两个（水平）矩形的任意一边是否有间距，从而得之两个矩形块有没有发生碰撞。具体实现方式，可以选定一个矩形为参照物，计算另一矩形的与自己相近的边是否发生重合现象。若四边均未发生重合，则未发生碰撞，反之则发生碰撞。

图形示例：



简单算法实现(非碰撞情况，else 分支就是碰撞情况)：

```
if( domA.left > domB.right
    || domA.top > domB.bottom
    || domA.right < domB.left
    || domA.bottom < domB.top )
{
    return false // 未碰撞
} else {
    return true // 碰撞
}
```

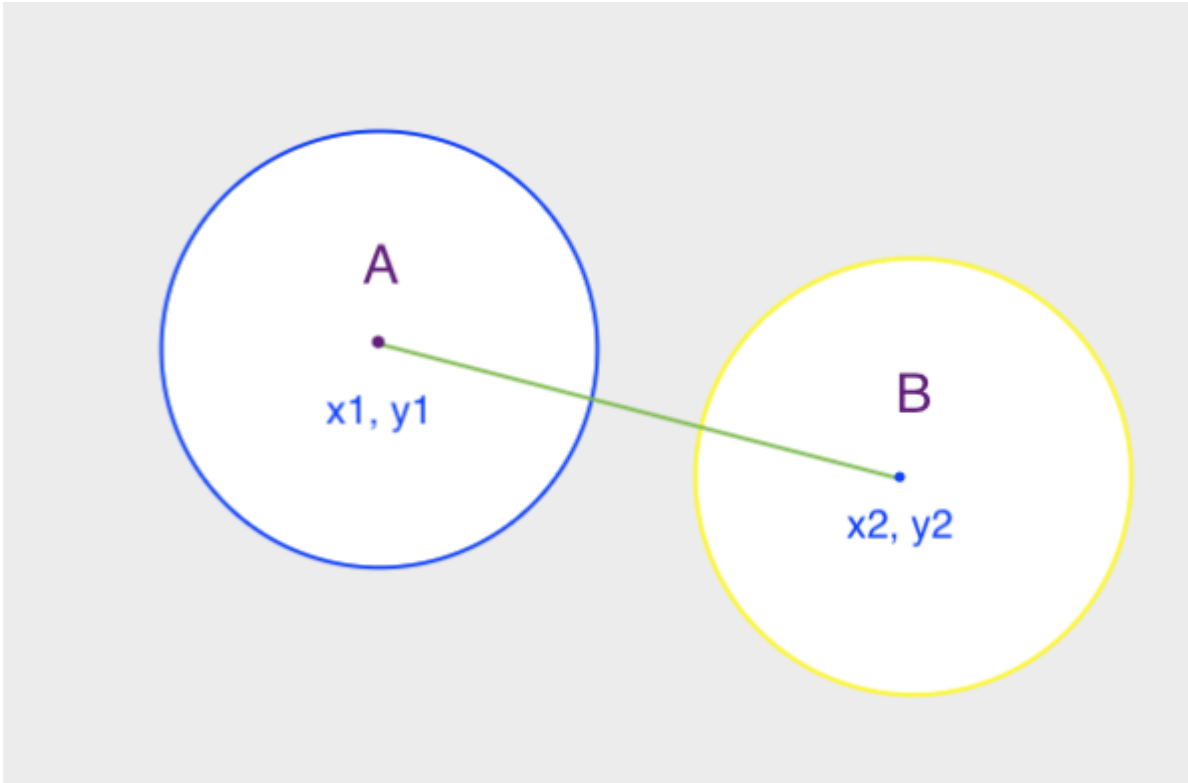
另一种实现

```
if(domA.left >)
```

圆形与圆形的碰撞：

判断任意两个圆形碰撞比较简单，只需要判断两个圆的圆心距离是否小于两圆半径之和，如果小于半径和，就可以判定两个圆发生碰撞。

图形示例：



简单算法实现

```
var distance = Math.sqrt(Math.pow(x1 - x2) + Math.pow(y1 - y2) )
if (distance < r1 + r2) { // r1、r2 分别为两圆的半径
    return true // 发生碰撞
} else {
    return false //未发生碰撞
}
```

防抖 节流

> 在前端开发的过程中，我们经常会需要绑定一些持续触发的事件，如 resize、scroll、mousemove 等等，但有些时候我们并不希望在事件持续触发的过程中那么频繁地去执行函数。

防抖 (debounce)

> 所谓防抖，就是指触发事件后在 n 秒内函数只能执行一次，如果在 n 秒内又触发了事件，则会重新计算函数执行时间。
>
> ps: 重置普攻

策略:
当事件被触发时，设定一个周期延时执行动作，若周期又被触发，则重新设定周期，直到周期结束，执行动作。
在后期有拓展了前缘防抖函数，即执行动作在前，设定延迟周期在后，周期内有事件被触发，不执行动作，且周期重新设定。

```
var oInput = $('.input-box');
var oShow = $('.show-box');
var timeOut;
oInput.addEventListener('input', function () {
    timeOut && clearTimeout(timeOut);
    timeOut = setTimeout(function () {
        oShow.innerText = translate(oInput.innerText);
    }, 500);
}, false);

function translate(str) {
    return str.split("").reverse().join("");
}
```

节流 (throttling)

> **所谓节流，就是指连续触发事件但是在 n 秒中只执行一次函数。**节流会稀释函数的执行频率。
>
> 对于节流，有多种方式可以实现 时间戳 定时器 束流等。
>
> ps : 技能CD

策略:
固定周期内，只执行一次动作，若没有新事件触发，不执行。周期结束后，又有事件触发，开始新的周期。
特点:
连续高频触发事件时，动作会被定期执行，响应平滑

计时器版

```
var oCon = $('.container');
var num = 0;
var valid = true;
oCon.addEventListener('mousemove', function () {
    if (!valid) {
        return false;
    }
    valid = false;
    setTimeout(function () {
        count();
        valid = true;
    }, 500);
}, false);
```

```
function count() {
    oCon.innerText = num++;
}
```

时间戳版

```
var oCon = $('.container');
var num = 0;
var time = Date.now();
oCon.addEventListener('mousemove', function () {
    if (Date.now() - time < 600) {
        return false;
    }
    time = Date.now();
    count();
}, false);

function count() {
    oCon.innerText = num++;
}
```

束流器版

```
var oCon = $('.container');
var num = 0;
var time = 0;
oCon.addEventListener('mousemove', function () {
    time++;
    if (time % 30 !== 0) {
        return false;
    }
    console.log(time)
    count();
}, false);

function count() {
    oCon.innerText = num++;
}
```

mousewheel事件

当用户通过鼠标滚轮与页面交互、在垂直方向上滚动页面时，就会触发mousewheel事件，这个事件就是实现全屏切换效果需要用到的。在IE6, IE7, IE8, Opera 10+, Safari 5+中，都提供了 “mousewheel” 事件，而 Firefox 3.5+ 中提供了一个等同的事件：”DOMMouseScroll”。与mousewheel事件对应的event对象中我们还会用到另一个特殊属性—wheelDelta属性。

1. “mousewheel” 事件中的 “event.wheelDelta” 属性值：返回的值，如果是正值说明滚轮是向上滚动，如果是负值说明滚轮是向下滚动；返回的值，均为 120 的倍数，即：幅度大小 = 返回的值 / 120。
2. “DOMMouseScroll” 事件中的 “event.detail” 属性值：返回的值，如果是负值说明滚轮是向上滚动（与 “event.wheelDelta” 正好相反），如果是正值说明滚轮是向下滚动；返回的值，均为 3 的倍数，即：幅度大小 = 返回的值 / 3。

封装

```
function mousewheel(obj,fn){

   obj.onmousewheel===null ? obj.onmousewheel=fun : obj.addEventListener('DOMMouseScroll',fun,false);

   function fun(e){
      var e=e || event,
         num=0;

      if(e.wheelDelta){
         num=e.wheelDelta>0?1:-1;
      }else{
         num=e.detail<0?1:-1;
      }
      fn(num);

      if(e.preventDefault)e.preventDefault();
      return false;
   }
}

//调用
var oDiv=document.getElementById('div');

mousewheel(oDiv,function(dir){
   if(dir>0) alert('向上滚动');
   if(dir<0) alert('往下滚动');
});
```

scrollIntoView()

> ele.`scrollIntoView()` 让元素滚动到可视区域（HTML5标准),参数 `true` 与浏览器对齐， `false` 元素在窗口居中显示

??运算符 (试验中的 运算符) (了解)

> ECMA 2021新增的试验 运算符 用来替代 || 在参数初始化等场景的使用 , 因为 || 在参数初始化 使用会对
>
> '' 0 false NaN null 等隐式转换中 会默认转换为 false的实参生效 导致实参被修改 无法准确达到 使用|| 做参数初始化的目的 (用户如果没有传入对应实参 参数设置为 xxx)
>
> 注意事项: ?? 不能和 || 或者 && 在同一行使用 如果非要在同一行代码使用 需要用 ( ) 严格括起来所有的运算短语

```
param = param || 0;

param = param ?? 0;

(param && slid) ?? (x || 0)
```

DOM基础操作导图


JavaScript
DOM基本操作
July
获取节点
document
getElementById
语法
document.getElementById(元素ID)
功能
通过元素ID获取节点
getElementsByName
语法
document.getElementsByName(元素name属性)
功能
通过元素的name属性获取节点
getElementsByTagName
语法
document.getElementsByTagName(元素标签)
功能
通过元素标签获取节点
节点指针
firstChild
语法
父节点.firstChild
功能
获取元素的首个子节点
lastChild
语法
父节点.lastChild
功能
获取元素的最后一个子节点
childNodes
语法
父节点.childNodes
功能
获取元素的子节点列表
previousSibling
语法
兄弟节点.previousSibling
功能
获取已知节点的前一个节点
nextSibling
语法
兄弟节点.nextSibling
功能
获取已知节点的后一个节点
parentNode
语法
子节点.parentNode
功能
获取已知节点的父节点
节点操作
创建节点
createElement
语法
document.createElement(元素标签)
功能
创建元素节点
createAttribute
语法
document.createAttribute(元素属性)
功能
创建属性节点
createTextNode
语法
document.createTextNode(文本内容)
功能
创建文本节点
插入节点
appendChild
语法
appendChild(所添加的新节点)
功能
向节点的子节点列表的末尾添加新的子节点
insertBefore
语法
insertBefore(所要添加的新节点,已知子节点)
功能
在已知的子节点前插入一个新的子节点
替换节点
replaceChild
语法
replaceChild(要插入的新元素,将被替换的老元素)
功能
将某个子节点替换为另一个
复制节点
cloneNode
语法
需要被复制的节点.cloneNode(true/false)
功能
创建指定节点的副本
参数
true
复制当前节点及其所有子节点
false
仅复制当前节点
删除节点
removeChild
语法
removeChild(要删除的节点)
功能
删除指定的节点
获取属性
getAttribute
语法
元素节点.getAttribute(元素属性名)
功能
获取元素节点中指定属性的属性值

BOM

Browser Object Mode， 浏览器对象模型。没有标准，浏览器厂家约定俗成。

BOM的核心是window，而window对象又具有双重角色，它既是通过js访问浏览器窗口的一个接口，又是一个Global（全局）对象。这意味着在网页中定义的任何对象，变量和函数，都以window作为其global对象。

BOM对象包括

- Window对象：浏览器中打开的窗口， 顶层对象
- Navigator对象：浏览器的相关信息
- Screen对象：客户端显示屏幕的信息
- History对象：用户在浏览器窗口中访问过的URL
- Location对象：当前URL的信息

> 其中，window对象中包含对BOM其他四个对象的只读引用以及Document对象的只读引用

document对象

document对象：实际上是window对象的属性

document == window.document为true，是唯一 一个既属于BOM又属于DOM的对象

document.lastModified

> 获取最后一次修改页面的日期的字符串表示

document.referrer

> 用于跟踪用户从哪里链接过来的

document.title

> 获取当前页面的标题，可读写

document.URL

> 获取当前页面的URL，可读写

document.anchors[0] / document.anchors["anchName"]

> 访问页面中所有的锚

document.forms[0] / document.forms["formName"]

> 访问页面中所有的表单

document.images[0] / document.images["imgName"]

> 访问页面中所有的图像

document.links [0] / document.links["linkName"]

> 访问页面中所有的链接

document.applets [0] / document.applets["appletName"]

> 访问页面中所有的Applet

document.embeds [0] / document.embeds["embedName"]

> 访问页面中所有的嵌入式对象

document.write(); / document.writeln();

> 将字符串插入到调用它们的位置

location对象

表示载入窗口的URL，也可用window.location引用它

location.href

> 当前载入页面的完整URL，如http://www.somewhere.com/pictures/index.htm

location.protocol

> URL中使用的协议，即双斜杠之前的部分，如http

location.host

> 服务器的名字，如www.wrox.com location.hostname //通常等于host，有时会省略前面的www

location.port

> URL声明的请求的端口，默认情况下，大多数URL没有端口信息，如8080

location.pathname

> URL中主机名后的部分，如/pictures/index.htm location.search //执行GET请求的URL中的问号后的部分，又称查询字符串，如?param=xxxx location.hash //如果URL包含#，返回该符号之后的内容，如#anchor1

location.assign("http:www.baidu.com");

> 同location.href，新地址都会被加到浏览器的历史栈中

location.replace("http:www.baidu.com");

> 同assign()，但新地址不会被加到浏览器的历史栈中，不能通过back和forward访问

location.reload(true | false);

> 重新载入当前页面，为false时从浏览器缓存中重载，为true时从服务器端重载，默认为false

navigator对象

`navigator`对象：

包含大量有关Web浏览器的信息，在检测浏览器及操作系统上非常有用，也可用window.navigator引用它

navigator.appCodeName

> 浏览器代码名的字符串表示 (一般都是Mozilla)

navigator.appName

> 官方浏览器名的字符串表示 一般都是 Netscape(网景)

navigator.appVersion

> 浏览器版本信息的字符串表示

navigator.cookieEnabled

> 如果启用cookie返回true，否则返回false

navigator.javaEnabled

> 如果启用java返回true，否则返回false

navigator.platform

> 浏览器所在计算机平台的字符串表示

navigator.plugins

> 安装在浏览器中的插件数组

navigator.taintEnabled

> 如果启用了数据污点返回true，否则返回false

navigator.userAgent

> 用户代理头的字符串表示

screen对象

screen对象：用于获取某些关于用户屏幕的信息，也可用window.screen引用它

screen.width/height

> 屏幕的宽度与高度，以像素计

screen.availWidth/availHeight

> 窗口可以使用的屏幕的宽度和高度，以像素计

screen.colorDepth

> 用户表示颜色的位数，大多数系统采用32位

window.moveTo(0, 0); window.resizeTo(screen.availWidth, screen.availHeight);

> 填充用户的屏幕 (失效 : 用户安全和隐私协议)

history 历史记录

History 对象包含用户（在浏览器窗口中）访问过的 URL。

`length` 返回浏览器历史列表中的 URL 数量。

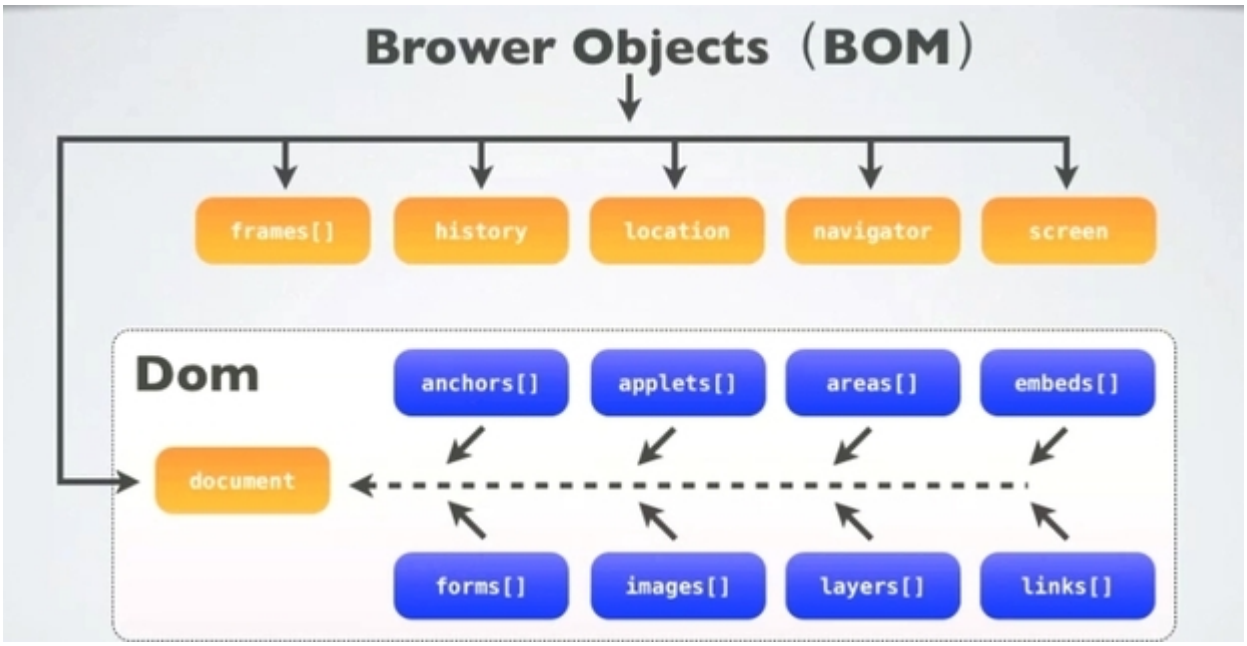
方法
`back()` 加载 history 列表中的前一个 URL。
`forward()` 加载 history 列表中的下一个 URL。
`go()` 加载 history 列表中的某个具体页面。
下面一行代码执行的操作与单击两次后退按钮执行的操作一样：

```
history.go(-2)
```

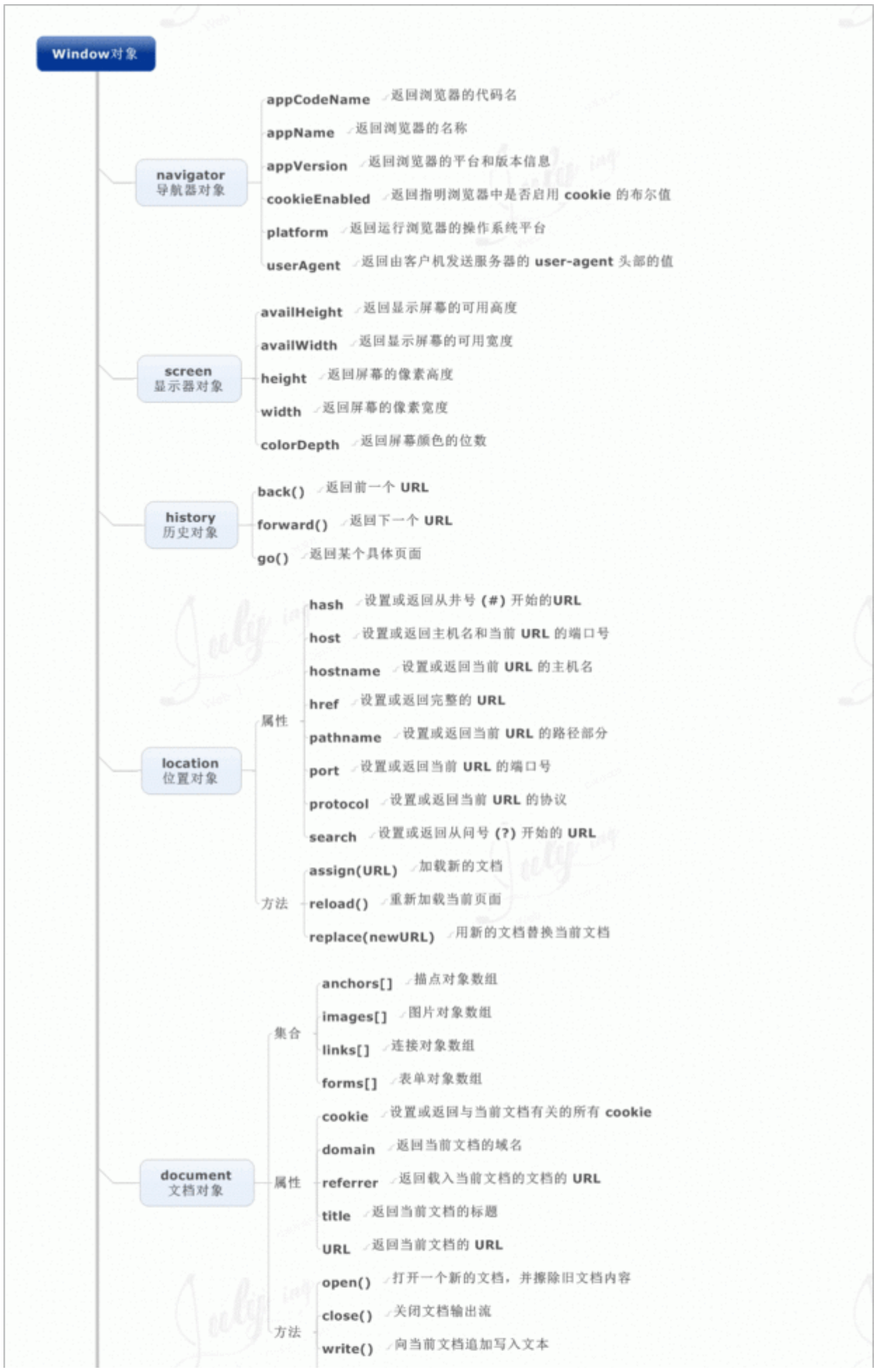
BOM 浏览器方法图表

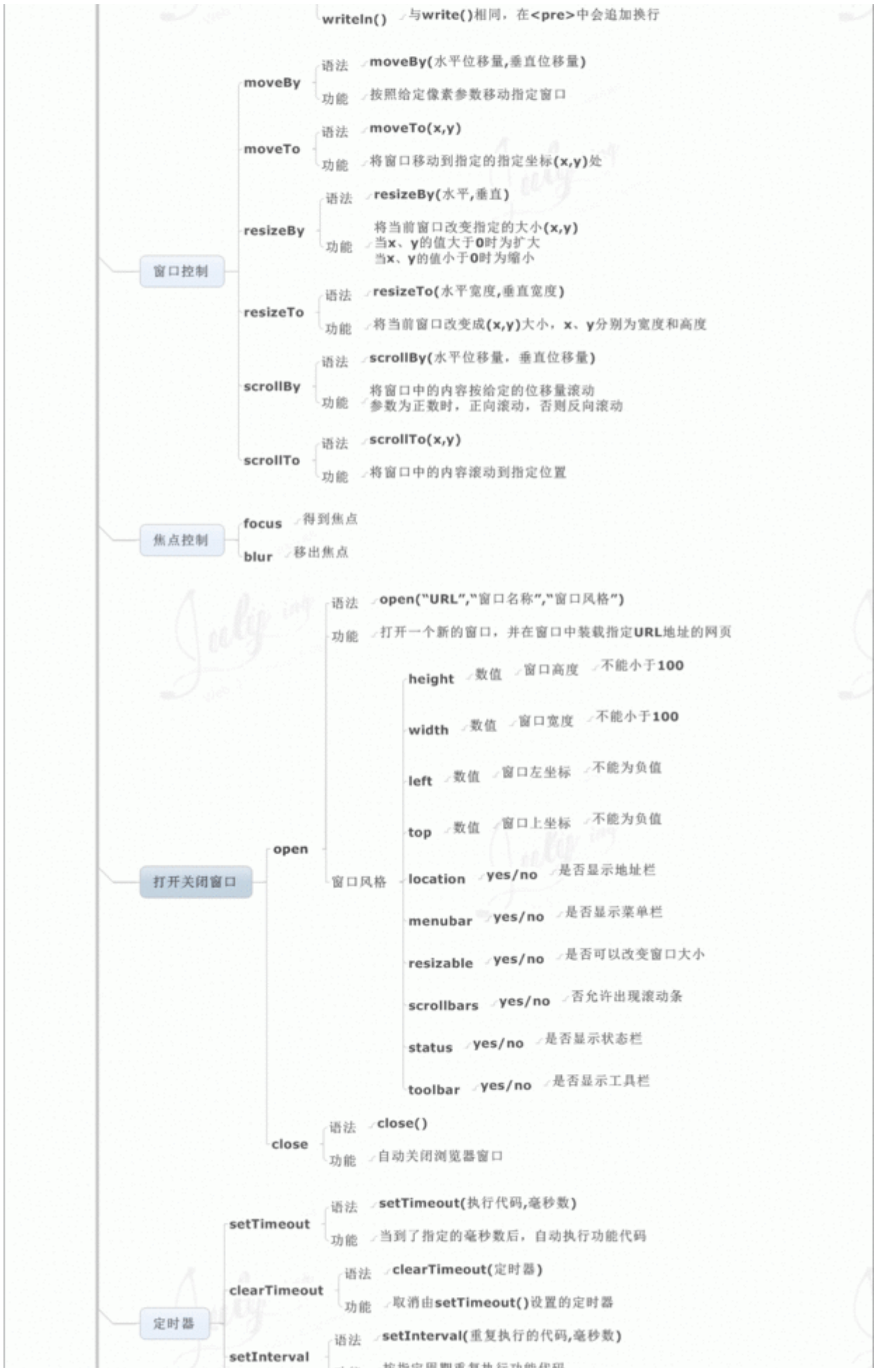
方法	描述
alert()	显示带有一段消息和一个确认按钮的警告框。
prompt()	显示可提示用户输入的对话框。
confirm()	显示带有一段消息以及确认按钮和取消按钮的对话框。
open()	打开一个新的浏览器窗口或查找一个已命名的窗口。
close()	关闭浏览器窗口。
print()	打印当前窗口的内容。
focus()	把键盘焦点给予一个窗口。
blur()	把键盘焦点从顶层窗口移开。
moveBy()	可相对窗口的当前坐标把它移动指定的像素。
moveTo()	把窗口的左上角移动到一个指定的坐标。
resizeBy()	按照指定的像素调整窗口的大小。
resizeTo()	把窗口的大小调整到指定的宽度和高度。
scrollBy()	按照指定的像素值来滚动内容。
scrollTo()	把内容滚动到指定的坐标。
setInterval()	每隔指定的时间执行代码。
setTimeout()	在指定的延迟时间之后来执行代码。
clearInterval()	取消 setInterval() 的设置。
clearTimeout()	取消 setTimeout() 的设置。

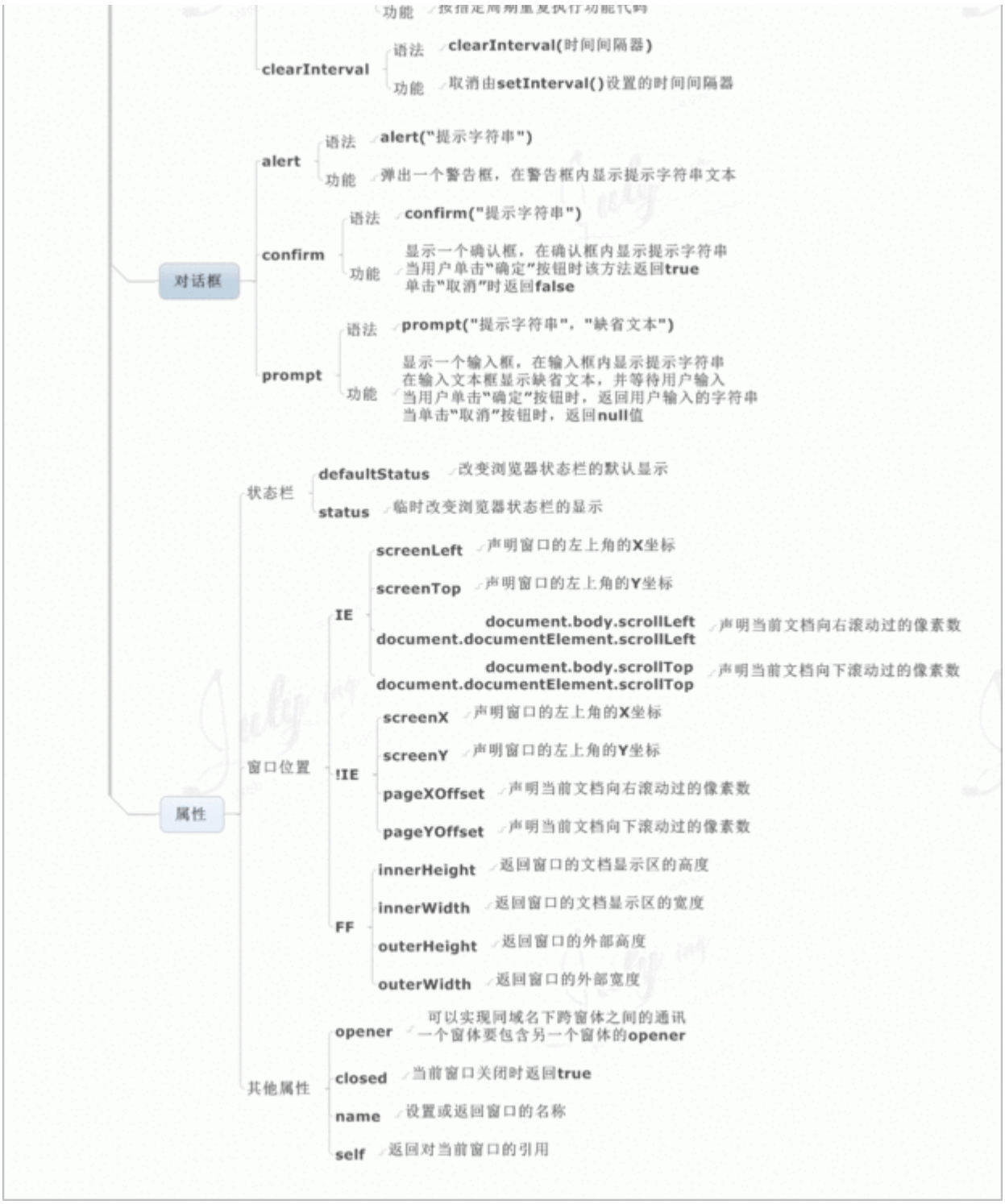
BOM和DOM的结构关系示意图



BOM导图


Window对象
navigator
导航器对象
appCodeName 返回浏览器的代码名
appName 返回浏览器的名称
appVersion 返回浏览器的平台和版本信息
cookieEnabled 返回指明浏览器中是否启用 cookie 的布尔值
platform 返回运行浏览器的操作系统平台
userAgent 返回由客户机发送服务器的 user-agent 头部的值
screen
显示器对象
availHeight 返回显示屏幕的可用高度
availWidth 返回显示屏幕的可用宽度
height 返回屏幕的像素高度
width 返回屏幕的像素宽度
colorDepth 返回屏幕颜色的位数
history
历史对象
back() 返回前一个 URL
forward() 返回下一个 URL
go() 返回某个具体页面
location
位置对象
属性
hash 设置或返回从井号 (#) 开始的URL
host 设置或返回主机名和当前 URL 的端口号
hostname 设置或返回当前 URL 的主机名
href 设置或返回完整的 URL
pathname 设置或返回当前 URL 的路径部分
port 设置或返回当前 URL 的端口号
protocol 设置或返回当前 URL 的协议
search 设置或返回从问号 (?) 开始的 URL
方法
assign(URL) 加载新的文档
reload() 重新加载当前页面
replace(newURL) 用新的文档替换当前文档
document
文档对象
集合
anchors[] 描点对象数组
images[] 图片对象数组
links[] 连接对象数组
forms[] 表单对象数组
属性
cookie 设置或返回与当前文档有关的所有 cookie
domain 返回当前文档的域名
referrer 返回载入当前文档的文档的 URL
title 返回当前文档的标题
URL 返回当前文档的 URL
方法
open() 打开一个新的文档，并擦除旧文档内容
close() 关闭文档输出流
write() 向当前文档追加写入文本


writeln() 与write()相同，在<pre>中会追加换行
窗口控制
moveBy
语法 moveBy(水平位移量,垂直位移量)
功能 按照给定像素参数移动指定窗口
moveTo
语法 moveTo(x,y)
功能 将窗口移动到指定的指定坐标(x,y)处
resizeBy
语法 resizeBy(水平,垂直)
功能 将当前窗口改变指定的大小(x,y)
当x、y的值大于0时为扩大
当x、y的值小于0时为缩小
resizeTo
语法 resizeTo(水平宽度,垂直宽度)
功能 将当前窗口改变成(x,y)大小，x、y分别为宽度和高度
scrollBy
语法 scrollBy(水平位移量，垂直位移量)
功能 将窗口中的内容按给定的位移量滚动
参数为正数时，正向滚动，否则反向滚动
scrollTo
语法 scrollTo(x,y)
功能 将窗口中的内容滚动到指定位置
焦点控制
focus 得到焦点
blur 移出焦点
打开关闭窗口
open
语法 open("URL","窗口名称","窗口风格")
功能 打开一个新的窗口，并在窗口中装载指定URL地址的网页
窗口风格
height 数值 窗口高度 不能小于100
width 数值 窗口宽度 不能小于100
left 数值 窗口左坐标 不能为负值
top 数值 窗口上坐标 不能为负值
location yes/no 是否显示地址栏
menubar yes/no 是否显示菜单栏
resizable yes/no 是否可以改变窗口大小
scrollbars yes/no 否允许出现滚动条
status yes/no 是否显示状态栏
toolbar yes/no 是否显示工具栏
close
语法 close()
功能 自动关闭浏览器窗口
定时器
setTimeout
语法 setTimeout(执行代码,毫秒数)
功能 当到了指定的毫秒数后，自动执行功能代码
clearTimeout
语法 clearTimeout(定时器)
功能 取消由setTimeout()设置的定时器
setInterval
语法 setInterval(重复执行的代码,毫秒数)



RegExp

什么是正则表达式

> 正则表达式是描述字符模式的对象。

- 正则表达式用于对字符串模式匹配及检索替换，是对`字符串`执行模式匹配的强大工具。
- 而`String`和`RegExp`都定义了使用正则表达式进行强大的模式匹配和文本检索与替换的函数。
- 正则表达式主要用来验证客户端的输入数据。可以节约大量的服务器端的系统资源，并且提供更好的用户体验。

创建正则表达式

1、直接量

语法：Reg = /pattern/modifiers;

```
var Reg = /box/gi;
```

2、new RegExp

语法 Reg = `new RegExp(` pattern , modifiers `)`;
pattern ， modifiers此时是`字符串`

```
var Reg = new RegExp("box","gi");
```

- 何种方法创建都是一样的
- pattern `模式 模板`，要匹配的内容
- modifiers `修饰符`

正则表达式用法及区别

1、String中正则表达式方法

方法	描述
`match( Reg )`	返回RegExp匹配的包含全部`字符串`的数组或`null`
`search( Reg )`	返回RegExp匹配字符串首次出现的`位置`
`replace( Reg，newStr )`	用`newStr`替换RegExp匹配结果，并返回新字符串
`split( Reg )`	返回字符串按指定RegExp拆分的数组

使用

```
var str = 'hello';
var Reg = /e/i;
str.match(Reg);
```

2、RegExp对象的方法

方 法

方法	描述
exec（）	在字符串中执行匹配搜索，返回首次匹配结果数组，
test（）	在字符串中测试模式匹配，返回 true 或 false

使用

```
var pattern = new RegExp("box","gi");
pattern.test(str);
pattern.exec(str);
```

> 注意区别正则方法和字符串方法使用避免混淆

正则方法：pattern.test(str) ; 方法的主体是正则表达式
字符串方法：str.match(pattern) ;方法的主体是字符串

修饰符

> 修饰符用于执行区分大小写和全局匹配:

- i 忽略大小写匹配
- g 全局匹配，默认只匹配第一个元素，就不在进行匹配
- m 执行多行匹配

```
var patt =  /pattern/i;       //忽略大小写匹配
var patt =  /pattern/g;       //全局匹配
var patt =  /pattern/m;       //执行多行匹配
```

---

pattern 模式

1、基本匹配

xxx ——————————————匹配xxx字符

```
var Reg = /abc/;
```

`x|y|z` ————————————-匹配`x`或`y`或`z`字符

```
var Reg = /abc|bac|cba/;
```

2、[]

`[abc]`————————————————-匹配abc之中的`任何一个`字符
非
`[^abc]`————————————————匹配`非`a`非`b`非`c字符的
到

```
[0-9]` ————————————————匹配`0至9`之间的数字
`[a-z]` ————————————————匹配`小写a至小写z`的字符
`[A-Z]` ————————————————匹配`大写A至大写Z`的字符
匹配中文 `[\u4e00-\u9fa5]
```

还可以组合

```
var Reg  = /hello [0-9a-zA-z]/;
```

元字符(`转义字符`)

```
.` ————————————————————匹配`单个字符`， 除了换行和行结束符
`\w`————————————————————匹配`单词字符`,`数字`,`_`(下划线)
`\W`————————————————————匹配`非` （`单词字符`和`_`(下划线))
`\d` ————————————————————匹配`数字`
`\D` ————————————————————匹配`非数字`
`\s` ————————————————————匹配空白字符 （`空格`）
`\S` ————————————————————匹配`非空格`字符
`\b` ————————————————————匹配`单词边界`（除了 (字)字母 数字_ 都算单词边界)
`\B` ————————————————————匹配`非单词边界`
`\n` ————————————————————匹配`换行符
```

特殊的转译字符`.` `\` `/`

```
var reg = /\./;//匹配.
var reg = /\\/;//匹配\
var reg = /\//;//匹配/
```

4、量词

`n?`————————————————匹配`0个或一个`n的字符串
`n*`————————————————匹配`0个或多个`字符串(`任意个`)
`n+`————————————————匹配`至少一个`n字符串
`n{X}`————————————————匹配包含`X个n`的序列的字符串
`n{X,Y}`————————————-匹配包含`至少X或至多Y个`n的序列的字符串
`n{x,}`————————————-匹配`至少X个`n的序列字符串

`^n`————————————————匹配`以n开头`的字符串
`n$`————————————————匹配`以n结尾`的字符串

5、贪婪惰性

> **贪婪**: `尽可能多`的匹配
> **惰性**: `尽可能少`的匹配
> 前提条件都是要匹配到内容

—— 贪婪 ——	—— 惰性 ——
`+`	`+?`
`?`	`??`
`*`	`*?`
`{n}`	`{n}?`
`{n,m}`	`{n,m}?`
`{n,}`	`{n,}?`

6、子组(子表达式)

子组:使用`()`小括号,指定一个子表达式后,称之为分组

- `捕获型`
- `非捕获型`

1）、捕获型

```
var str = 'abcdefg';
var reg = /(abc)d/;//匹配abcd
var val = reg.exec( str);
console.log( val );   //["abcd", "abc", index: 0, input: "abcdefg"]
```

索引0 为匹配的结果
索引1 为第一个子表达式 匹配结果
index :首次匹配成功的索引值,
input: **匹配目标**

—— 字符 ——		引用
(pattern)	匹配pattern并`捕获结果`， `自动设置组号`,是从1开始的正整数	\num

引用是`值的引用，匹配结果的引用`不是匹配形式引用

1）、非捕获型

- (? :pattern)
- (?=pattern) 零宽度正向预言

```
Windows (?=2000) //匹配windows且后面跟2000
```

匹配 “Windows2000” 中的 “Windows”
不匹配 “Windows3.1” 中的 “Windows”。

- (?!pattern) 零宽度负向预言

```
Windows (?!2000)//匹配windows且后面非2000
```

匹配 “Windows3.1” 中的 “Windows”
不匹配 “Windows2000” 中的 “Windows”。

常用正则

火车车次

```
/^[GCDZTSPKXLY1-9]\d{1,4}$/
```

手机机身码(IMEI)

```
/^\d{15,17}$/
```

必须带端口号的网址(或ip)

```
/^((ht|f)tps?:\/\/)?[\w-]+(\.[\w-]+)+:\d{1,5}\/?$/
```

网址(url,支持端口和"?+参数"和"#+参数)

```
/^(((ht|f)tps?):\/\/)?[\w-]+(\.[\w-]+)+([\w.,@?^=%&:/~+#-]*[\w@?^=%&/~+#-])?$/
```

统一社会信用代码

```
/^[0-9A-HJ-NPQRTUWXY]{2}\d{6}[0-9A-HJ-NPQRTUWXY]{10}$/
```

迅雷链接

```
/^thunderx?:\/\/[a-zA-Z\d]+=$/
```

ed2k链接(宽松匹配)

```
/^ed2k:\/\/\|file\|.+\|\/$/
```

磁力链接(宽松匹配)

```
/^magnet:\?xt=urn:btih:[0-9a-fA-F]{40,}.*$/
```

子网掩码

```
/^(?:\d{1,2}|1\d\d|2[0-4]\d|25[0-5])(?:\.(?:\d{1,2}|1\d\d|2[0-4]\d|25[0-5])){3}$/
```

linux"隐藏文件"路径

```
/^\/(?:[^/]+\/)*\.[^/]*/
```

linux文件夹路径

```
/^\/(?:[^/]+\/)*$/
```

linux文件路径

```
/^\/(?:[^/]+\/)*[^/]+$/
```

window"文件夹"路径

```
/^[a-zA-Z]:\\(?:\w+\\?)*$/
```

window下"文件"路径

```
/^[a-zA-Z]:\\(?:\w+\\)*\w+\.\w+$/
```

股票代码(A股)

```
/^(s[hz]|S[HZ])(000[\d]{3}|002[\d]{3}|300[\d]{3}|600[\d]{3}|60[\d]{4})$/
```

大于等于0, 小于等于150, 支持小数位出现5, 如145.5, 用于判断考卷分数

```
/^150$|^(?:\d|[1-9]\d|1[0-4]\d)(?:.5)?$/
```

html注释

```
/^<!--[\s\S]*?-->$/
```

md5格式(32位)

```
/^([a-f\d]{32}|[A-F\d]{32})$/
```

版本号(version)格式必须为X.Y.Z

```
/^\d+(?:\.\d+){2}$/
```

视频(video)链接地址（视频格式可按需增删）

```
/^https?:\/\/(.+\/)+.+(\.(swf|avi|flv|mpg|rm|mov|wav|asf|3gp|mkv|rmvb|mp4))$/i
```

图片(image)链接地址（图片格式可按需增删）

```
/^https?:\/\/(.+\/)+.+(\.(gif|png|jpg|jpeg|webp|svg|psd|bmp|tif))$/i
```

24小时制时间（HH:mm:ss）

```
/^(?:[01]\d|2[0-3]):[0-5]\d:[0-5]\d$/
```

12小时制时间（hh:mm:ss）

```
/^(?:1[0-2]|0?[1-9]):[0-5]\d:[0-5]\d$/
```

base64格式

```
/^\s*data:(?:[a-z]+\/[a-z0-9-+.]+(?:;[a-z-]+=[a-z0-9-]+)?)?(?:;base64)?,([a-z0-9!$&',()*+;=\-._~:@/?%\s]*?)\s*$/i
```

数字/货币金额（支持负数、千分位分隔符）

```
/^-?\d+(,\d{3})*(\.\d{1,2})?$/
```

数字/货币金额 (只支持正数、不支持校验千分位分隔符)

```
/(?:^[1-9]([0-9]+)?(?:\.[0-9]{1,2})?$)|(?:^(?:0){1}$)|(?:^[0-9]\.[0-9](?:[0-9])?$)/
```

银行卡号（10到30位, 覆盖对公/私账户, 参考微信支付）

```
/^[1-9]\d{9,29}$/
```

中文姓名

```
/^(?:[\u4e00-\u9fa5·]{2,16})$/
```

英文姓名

```
/(^[a-zA-Z]{1}[a-zA-Z\s]{0,20}[a-zA-Z]{1}$)/
```

车牌号(新能源)

```
/[京津沪渝冀豫云辽黑湘皖鲁新苏浙赣鄂桂甘晋蒙陕吉闽贵粤青藏川宁琼使领 A-Z]{1}[A-HJ-NP-Z]{1}(([0-9]{5}[DF])|([DF][A-HJ-NP-Z0-9][0-9]{4}))$/
```

车牌号(非新能源)

```
/^[京津沪渝冀豫云辽黑湘皖鲁新苏浙赣鄂桂甘晋蒙陕吉闽贵粤青藏川宁琼使领 A-Z]{1}[A-HJ-NP-Z]{1}[A-Z0-9]{4}[A-Z0-9挂学警港澳]{1}$/
```

车牌号(新能源+非新能源)

```
/^(?:[京津沪渝冀豫云辽黑湘皖鲁新苏浙赣鄂桂甘晋蒙陕吉闽贵粤青藏川宁琼使领 A-Z]{1}[A-HJ-NP-Z]{1}(?:(?:[0-9]{5}[DF])|(?:[DF](?:[A-HJ-NP-Z0-9])[0-9]{4})))|(?:[京津沪渝冀豫云辽黑湘皖鲁新苏浙赣鄂桂甘晋蒙陕吉闽贵粤青藏川宁琼使领 A-Z]{1}[A-Z]{1}[A-HJ-NP-Z0-9]{4}[A-HJ-NP-Z0-9 挂学警港澳]{1})$/
```

手机号(mobile phone)中国(严谨), 根据工信部2019年最新公布的手机号段

```
/^(?:(?:\+|00)86)?1(?:(?:3[\d])|(?:4[5-7|9])|(?:5[0-3|5-9])|(?:6[5-7])|(?:7[0-8])|(?:8[\d])|(?:9[1|8|9]))\d{8}$/
```

手机号(mobile phone)中国(宽松), 只要是13,14,15,16,17,18,19开头即可

```
/^(?:(?:\+|00)86)?1[3-9]\d{9}$/
```

手机号(mobile phone)中国(最宽松), 只要是1开头即可, 如果你的手机号是用来接收短信, 优先建议选择这一条

```
/^(?:(?:\+|00)86)?1\d{10}$/
```

date(日期)

```
/^\d{4}(-)(1[0-2]|0?\d)\1([0-2]\d|\d|30|31)$/
```

email(邮箱)

```
/^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/
```

座机(tel phone)电话(国内),如: 0341-86091234

```
/^\d{3}-\d{8}$|^\d{4}-\d{7,8}$/
```

身份证号(1代,15位数字)

```
/^[1-9]\d{7}(?:0\d|10|11|12)(?:0[1-9]|[1-2][\d]|30|31)\d{3}$/
```

身份证号(2代,18位数字),最后一位是校验位,可能为数字或字符X

```
/^[1-9]\d{5}(?:18|19|20)\d{2}(?:0[1-9]|10|11|12)(?:0[1-9]|[1-2]\d|30|31)\d{3}[\dXx]$/
```

身份证号, 支持1/2代(15位/18位数字)

```
/(^\d{8}(0\d|10|11|12)([0-2]\d|30|31)\d{3}$)|(^\d{6}(18|19|20)\d{2}(0[1-9]|10|11|12)([0-2]\d|30|31)\d{3}(\d|X|x)$)/
```

护照（包含香港、澳门)

```
/(^[EeKkGgDdSsPpHh]\d{8}$)|(^(([Ee][a-fA-F])|([DdSsPp][Ee])|([Kk][Jj])|([Mm][Aa])|(1[45]))\d{7}$)/
```

帐号是否合法(字母开头，允许5-16字节，允许字母数字下划线组合

```
/^[a-zA-Z]\w{4,15}$/
```

中文/汉字

```
/^(?:[\u3400-\u4DB5\u4E00-\u9FEA\uFA0E\uFA0F\uFA11\uFA13\uFA14\uFA1F\uFA21\uFA23\uFA24\uFA27-\uFA29]|[\uD840-\uD868\uD86A-\uD86C\uD86F-\uD872\uD874-\uD879][\uDC00-\uDFFF]|\uD869[\uDC00-\uDED6\uDF00-\uDFFF]|\uD86D[\uDC00-\uDF34\uDF40-\uDFFF]|\uD86E[\uDC00-\uDC1D\uDC20-\uDFFF]|\uD873[\uDC00-\uDEA1\uDEB0-\uDFFF]|\uD87A[\uDC00-\uDFE0])+$/
```

小数

```
/^\d+\.\d+$/
```

数字

```
/^\d{1,}$/
```

html标签(宽松匹配)

```
/<(\w+)[^>]*>(.*?<\/\1>)?/
```

qq号格式正确

```
/^[1-9][0-9]{4,10}$/
```

数字和字母组成

```
/^[A-Za-z0-9]+$/
```

英文字母

```
/^[a-zA-Z]+$/
```

小写英文字母组成

```
/^[a-z]+$/
```

大写英文字母

```
/^[A-Z]+$/
```

密码强度校验，最少6位，包括至少1个大写字母，1个小写字母，1个数字，1个特殊字符

```
/^\S*(?=\S{6,})(?=\S*\d)(?=\S*[A-Z])(?=\S*[a-z])(?=\S*[!@#$%^&*? ])\S*$/
```

用户名校验，4到16位（字母，数字，下划线，减号）

```
/^[a-zA-Z0-9_-]{4,16}$/
```

ip-v4

```
/^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$/
```

ip-v6

```
/^((([0-9A-Fa-f]{1,4}:){7}[0-9A-Fa-f]{1,4})|(([0-9A-Fa-f]{1,4}:){6}:[0-9A-Fa-f]{1,4})|(([0-9A-Fa-f]{1,4}:){5}:([0-9A-Fa-f]{1,4}:)?[0-9A-Fa-f]{1,4})|(([0-9A-Fa-f]{1,4}:){4}:([0-9A-Fa-f]{1,4}:){0,2}[0-9A-Fa-f]{1,4})|(([0-9A-Fa-f]{1,4}:){3}:([0-9A-Fa-f]{1,4}:){0,3}[0-9A-Fa-f]{1,4})|(([0-9A-Fa-f]{1,4}:){2}:([0-9A-Fa-f]{1,4}:){0,4}[0-9A-Fa-f]{1,4})|(([0-9A-Fa-f]{1,4}:){6}((\b((25[0-5])|(1\d{2})|(2[0-4]\d)|(\d{1,2}))\b)\.){3}(\b((25[0-5])|(1\d{2})|(2[0-4]\d)|(\d{1,2}))\b))|(([0-9A-Fa-f]{1,4}:){0,5}:((\b((25[0-5])|(1\d{2})|(2[0-4]\d)|(\d{1,2}))\b)\.){3}(\b((25[0-5])|(1\d{2})|(2[0-4]\d)|(\d{1,2}))\b))|(::([0-9A-Fa-f]{1,4}:){0,5}((\b((25[0-5])|(1\d{2})|(2[0-4]\d)|(\d{1,2}))\b)\.){3}(\b((25[0-5])|(1\d{2})|(2[0-4]\d)|(\d{1,2}))\b))|([0-9A-Fa-f]{1,4}::([0-9A-Fa-f]{1,4}:){0,5}[0-9A-Fa-f]{1,4})|(::([0-9A-Fa-f]{1,4}:){0,6}[0-9A-Fa-f]{1,4})|(([0-9A-Fa-f]{1,4}:){1,7}:))$/i
```

16进制颜色

```
/^#?([a-fA-F0-9]{6}|[a-fA-F0-9]{3})$/
```

微信号(wx)，6至20位，以字母开头，字母，数字，减号，下划线

```
/^[a-zA-Z][-_a-zA-Z0-9]{5,19}$/
```

邮政编码(中国)

```
/^(0[1-7]|1[0-356]|2[0-7]|3[0-6]|4[0-7]|5[1-7]|6[1-7]|7[0-5]|8[013-6])\d{4}$/
```

中文和数字

```
/^((?:[\u3400-\u4DB5\u4E00-\u9FEA\uFA0E\uFA0F\uFA11\uFA13\uFA14\uFA1F\uFA21\uFA23\uFA24\uFA27-\uFA29]|[\uD840-\uD868\uD86A-\uD86C\uD86F-\uD872\uD874-\uD879][\uDC00-\uDFFF]|\uD869[\uDC00-\uDED6\uDF00-\uDFFF]|\uD86D[\uDC00-\uDF34\uDF40-\uDFFF]|\uD86E[\uDC00-\uDC1D\uDC20-\uDFFF]|\uD873[\uDC00-\uDEA1\uDEB0-\uDFFF]|\uD87A[\uDC00-\uDFE0])|(\d))+$/
```

不能包含字母

```
/^[^A-Za-z]*$/
```

java包名

```
/^([a-zA-Z_][a-zA-Z0-9_]*)+([.][a-zA-Z_][a-zA-Z0-9_]*)+$/
```

mac地址

```
/^((([a-f0-9]{2}:){5})|(([a-f0-9]{2}-){5}))[a-f0-9]{2}$/i
```

匹配连续重复的字符

```
/(.)\1+/
```

cookie

cookie：`存储数据`，当用户访问了某个网站（网页）的时候，我们就可以通过cookie来向访问者电脑上存储数据

> 1.不同的浏览器存放的cookie位置不一样，也是不能通用的
> 2.cookie的存储是以`域名形式`进行区分的
> 3.cookie的数据可以设置名字的
> 4.一个域名下存放的cookie的个数是有限制的，不同的浏览器存放的个数不一样
> 5.每个cookie存放的内容大小也是有限制的，不同的浏览器存放大小不一样

访问cookie

要在服务器环境下

> 我们通过`document.cookie`来获取当前网站下的cookie的时候，得到的字符串形式的值，他包含了当前网站下所有的cookie。他会把所有的cookie通过一个`分号+空格`的形式串联起来

```
consol.log( document.cookie );
```

存储cookie

```
document.cookie = '数据名=值';
```

设置cookie过期时间

> `cookie默认是临时存储的`，当浏览器关闭进程的时候自动销毁 ,如果我们想长时间存放一个cookie。需要在设置这个cookie的时候同时给他设置一个过期的时间
>
> - 过期时间必须是一个日期对象转`换成的字符串`（时间戳.`toGMTString()`）

document.cookie = '数据名=值; `expires` =过期时间';

```
var oDate = new Date();
oDate.setDate( oDate.getDate() + 5);

oDate.toGMTString();//转换为日期字符串
document.cookie='age=20; expires='+oDate;


/*-- document.cookie='sex=man\n你好'; */
//转码
var content= encodeURI('man\n你好');
document.cookie='sex='+content+';expires='+oDate;
```

要找到对应的数据值，可以使用多种方式

cookie封装

- 设置cookie封装

```
function setCookie(obj,time){
   for(key in obj){
      var d = new Date();
      d.setDate( d.getDate()+time );
      document.cookie = key+'='+obj[key]+'; expires='+d.toUTCString();

   }
}

setCookie({
   name:'hello',

   sex:'man',
```

```
    love:'逛街',
    work:'future'
},5);
```

- 获取cookie封装

```
function getCookie() {
    var cookie = document.cookie;
    var cookieArr = cookie.match(/[^=;\s]+=([^=;]+)(?:;?)/g);
    var argData = {}
    for (var key of arguments) {
        argData[key] = 1;
    }
    return cookieArr.reduce(function (acc, curr) {
        var tempStr = curr.replace(';', '');
        var tempArr = tempStr.split('=');
        if (tempArr[0] && argData[tempArr[0]]) {
            acc[tempArr[0]] = tempArr[1]
        }
        return acc;
    }, {});
getCookie('name','age')
```

- 移除cookie

```
function removeCookie(){
        for(key in arguments){
            var json ={};
          json[arguments[key]]=null;
           setCookie(json,-1);
        }
      }
removeCookie('name');
```