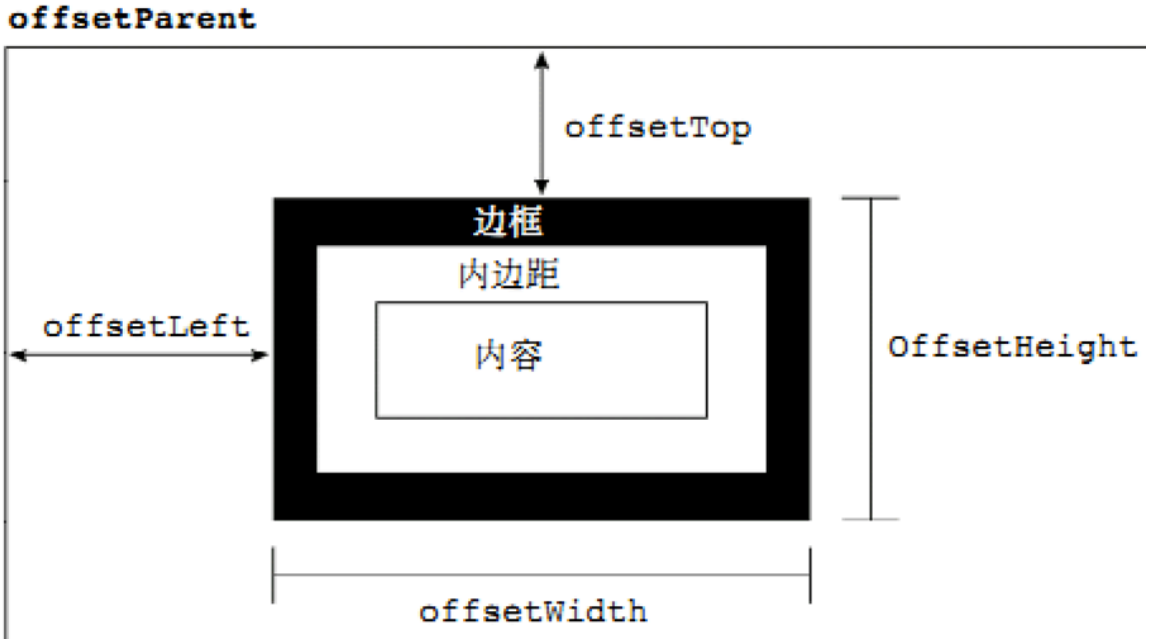
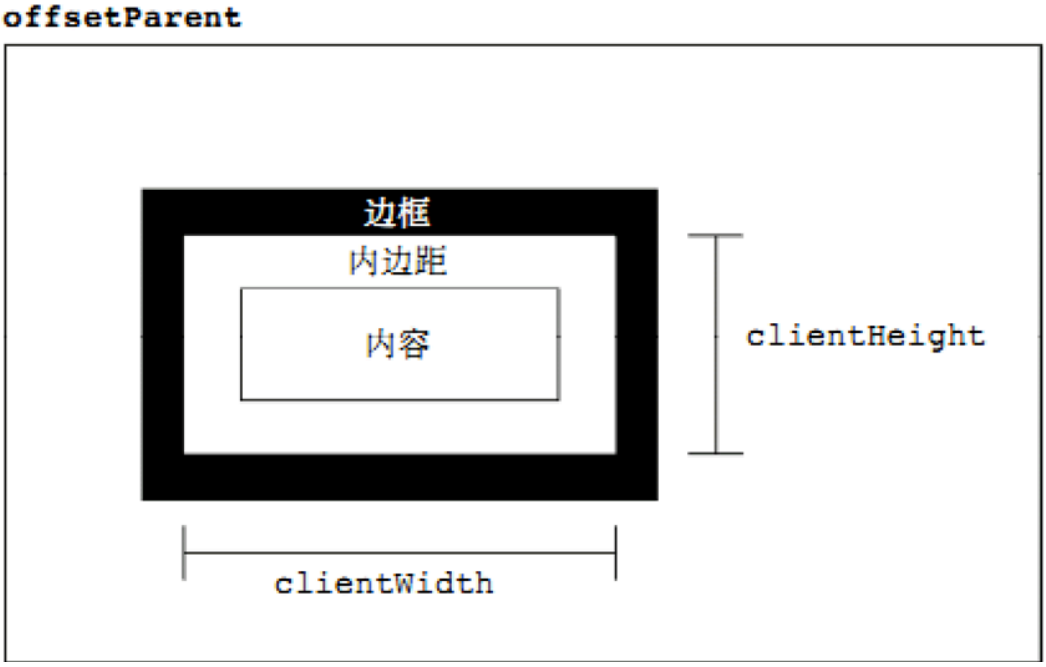
特效

- offsetParent用于获取定位的父级元素

offsetParent different from parentNode

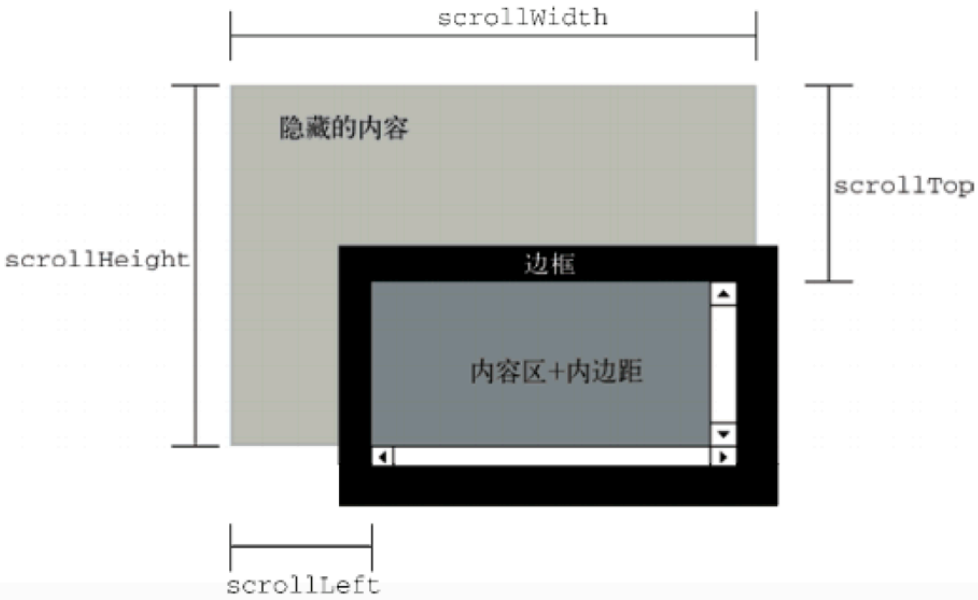


客户区



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)
```



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

阻止事件传播的方式

- 标准方式 event.stopPropagation();
- IE低版本 event.cancelBubble = true; 标准中已废弃

常用的鼠标和键盘事件

- onmouseup 鼠标按键放开时触发
- onmousedown 鼠标按键按下触发
- onmousemove 鼠标移动触发

- onkeyup 键盘按键按下触发
- onkeydown 键盘按键抬起触发

注册/移除事件的三种方式

```
var box = document.getElementById('box');
box.onclick = function () {
 console.log('点击后执行');
};
box.onclick = null;
//1
box.addEventListener('click', eventCode, false);
box.removeEventListener('click', eventCode, false);
//2
box.attachEvent('onclick', eventCode);
box.detachEvent('onclick', eventCode);
//3
function eventCode() {
 console.log('点击后执行');
}
```

兼容代码（兼容多个浏览器） better 用

```
function addEventListener(element, type, fn) {
 if (element.addEventListener) {
   element.addEventListener(type, fn, false);
} else if (element.attachEvent){
   element.attachEvent('on' + type,fn);
} else {
   element['on'+type] = fn;
}
}

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;
}
}
```

写入

创建元素的三种方式

document.write()

```
document.write('新设置的内容<p>标签也可以生成</p>');
```

innerHTML

```
var box = document.getElementById('box');
box.innerHTML = '新内容<p>新标签</p>';
```

document.createElement()

```
var div = document.createElement('div');
document.body.appendChild(div);
```

性能问题

- innerHTML方法由于会对字符串进行解析，需要避免在循环内多次使用。
- 可以借助字符串或数组的方式进行替换，再设置给innerHTML
- 优化后与document.createElement性能相近

最后再写入

Node

- **父节点（Parent Node）**：一个节点的父节点是直接包含该节点的节点。可以将父节点想象为节点的上一层，它是直接包围并包含该节点的节点。一个节点只能有一个父节点。
- **子节点（Child Nodes）**：一个节点的子节点是由该节点直接包含的节点。可以将子节点想象为节点的下一层，它们由父节点直接包围并位于其内部。一个节点可以有零个、一个或多个子节点。
- **兄弟节点（Sibling Nodes）**：兄弟节点是具有相同父节点的节点。它们是在同一层级下直接由同一个父节点包含的节点。兄弟节点在父节点的子节点列表中紧密相连。

下面是一个示例来说明这些属性之间的关系：

html Copy

```html
<body>
  <div id="parent">
    <p>子节点1</p>
    <p>子节点2</p>
    <p>子节点3</p>
  </div>
</body>
```

在这个示例中， 元素是 <p> 元素的父节点，而 <p> 元素是 元素的子节点。同时，这三个 <p> 元素是兄弟节点，它们都具有相同的父节点。

通过 JavaScript，我们可以使用 DOM 方法和属性来访问和操作节点层级关系。以下是一些示例：

javascript Copy

```javascript
var parentElement = document.getElementById('parent');
var childNodes = parentElement.childNodes; // 获取所有子节点
var firstChild = parentElement.firstChild; // 获取第一个子节点
var lastChild = parentElement.lastChild; // 获取最后一个子节点
var parentNode = childNodes[0].parentNode; // 获取父节点
var previousSibling = childNodes[1].previousSibling; // 获取前一个兄弟节点
```

节点操作，方法

appendChild()
insertBefore()
removeChild()
replaceChild()

节点层次，属性

parentNode
childNodes
children
nextSibling/previousSibling
firstChild/lastChild

表单元素属性

- value 用于大部分表单元素的内容获取(option除外)
- type 可以获取input标签的类型(输入框或复选框等)
- disabled 禁用属性
- checked 复选框选中属性
- selected 下拉菜单选中属性

样式操作

- 使用style方式设置的样式显示在标签行内

```
var box = document.getElementById('box');
box.style.width = '100px';
box.style.height = '100px';
box.style.backgroundColor = 'red';
```

- 注意 通过样式属性设置宽高、位置的属性类型是字符串，需要加上px

类名操作

- 修改标签的className属性相当于直接修改标签的类名

```
var box = document.getElementById('box');
box.className = 'clearfix';
```

案例

- 给文本框赋值，获取文本框的值
- 点击按钮禁用文本框
- 搜索文本框
- 检测用户名是否是3-6位，密码是否是6-8位，如果不满足要求高亮显示文本框
- 设置下拉框中的选中项
- 全选反选

自定义属性操作

- getAttribute() 获取标签行内属性
- setAttribute() 设置标签行内属性
- removeAttribute() 移除标签行内属性
- 与element.属性的区别: 上述三个方法用于获取任意的行内属性。

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)

非表单元素的属性

href、title、id、src、className

```
var link = document.getElementById('link');
console.log(link.href);
console.log(link.title);

var pic = document.getElementById('pic');
console.log(pic.src);
```

案例：

点击按钮，切换img标签里的图片

点击按钮显示隐藏div

- innerHTML和innerText

```
var box = document.getElementById('box');
box.innerHTML = '我是文本<p>我会生成为标签</p>';
console.log(box.innerHTML);
box.innerText = '我是文本<p>我不会生成为标签</p>';
console.log(box.innerText);
```

```
element.innerHTML = '<p>New content</p>'; // 设置元素的 HTML 内容
```

```
element.innerText = 'New content'; // 设置元素的纯文本内容
```

- HTML转义符

Similar to
" "
‘ '
& &
< < //less than 小于
> > // greater than 大于
空格
© ©

-

DOM经常进行的操作

- 获取元素
- 动态创建元素
- 对元素进行操作(设置其属性或调用其
- 事件(什么时机做相应的操作)

获取页面元素

##案例
1.点击按钮弹出对话框
2.点击按钮修改超链接的地址和热点文字
3.点击(每个)图片弹出对话框
4.点击图片设置自身宽和高
5.点击按钮修改每个图片的title属性
6.点击按钮显示哈哈(排他功能)
7.点击按钮显示和隐藏div
8.显示和隐藏二维码
9.点击按钮修改所有p标签内容
10.点击按钮修改所有文本框内容
11.点击按钮切换图片
12.点击超链接停止跳转页面
13.点击小图显示大图
14.美女相册
15点击按钮选中性别和兴趣

掌握
    getElementById()
    getElementsByTagName()。// eg 'div'
了解

getElementsByName()
getElementsByClassName()

选择器

querySelector() // ‘#text'
querySelectorAll() //'.box'

history对象

- back()
- forward()
- go()

navigator对象

- userAgent

通过userAgent可以判断用户浏览器的类型

- platform

通过platform可以判断浏览器所在的系统平台类型.

location对象 user resource location

```
> location
<- Location {ancestorOrigins: DOMStringList, href: 'https://poe.com/chat/2mbej98m2c95qhdf51f', origin: 'https://poe.com', protocol: 'https:', host: 'poe.com', …}
    ancestorOrigins: DOMStringList {length: 0}
    assign: ƒ assign()
    hash: ""
    host: "poe.com"
    hostname: "poe.com"
    href: "https://poe.com/chat/2mbej98m2c95qhdf51f"
    origin: "https://poe.com"
    pathname: "/chat/2mbej98m2c95qhdf51f"
    port: ""
    protocol: "https:"
    reload: ƒ reload()
    replace: ƒ replace()
    search: ""
    toString: ƒ toString()
    valueOf: ƒ valueOf()
    Symbol(Symbol.toPrimitive): undefined
    [[Prototype]]: Location
```

JS getUrlQuery.js 5

webAPI资料源码 > JS getUrlQuery.js > ...

```
1   function getQuery(queryStr) {
2       var query = {};
3       if (queryStr.indexOf('?') > -1) {
4         var index = queryStr.indexOf('?');
5         queryStr = queryStr.substr(index + 1);
6         var array = queryStr.split('&');
7         for (var i = 0; i < array.length; i++) {
8           var tmpArr = array[i].split('=');
9           if (tmpArr.length === 2) {
10            query[tmpArr[0]] = tmpArr[1];
11          }
12        }
13      }
14      return query;
15    }
16    console.log(getQuery(location.search));
17    console.log(getQuery(location.href));
```

```
> location
```

```
Location {ancestorOrigins: DOMStringList, href: 'https://www.google.com/search?q=search&oq=search&g…80gEIMjIzOWowajSoAgCwAgA&sourceid=chrome&ie=UTF-8', origin: 'https://www.google.com', protocol: 'https:', host: 'www.google.com', …}
  ancestorOrigins: DOMStringList {length: 0}
  assign: ƒ assign()
  hash: ""
  host: "www.google.com"
  hostname: "www.google.com"
  href: "https://www.google.com/search?q=search&oq=search&gs_lcrp=EgZjaHJ
  origin: "https://www.google.com"
  pathname: "/search"
  port: ""
  protocol: "https:"
  reload: ƒ reload()
  replace: ƒ replace()
  search: "?q=search&oq=search&gs_lcrp=EgZjaHJvbWUyCwgAEEUYDBg5GIAEMgkIAR
  toString: ƒ toString()
  valueOf: ƒ valueOf()
  Symbol(Symbol.toPrimitive): undefined
  [[Prototype]]: Location
```

```
> location.search
```

```
'?q=search&oq=search&gs_lcrp=EgZjaHJvbWUyCwgAEEUYDBg5GIAEMgkIARAAGAwYgAQyCQgCEAAYDBiABDIHCAMQABiABDIJCAQQABgMGIAEMgYIBRBFGDwyBggGEEUYPDIGCAcQRRg80gEIMjIzOWowajSoAgCwAgA&sourceid=chrome&ie=UTF-8'
```

```
> location.href
```

```
'https://www.google.com/search?q=search&oq=search&gs_lcrp=EgZjaHJvbWUyCwgAEE…BFGDwyBggGEEUYPDIGCAcQRRg80gEIMjIzOWowajSoAgCwAgA&sourceid=chrome&ie=UTF-8'
```

```
>
```

```
> getQuery(location.search)
```

```
{q: 'search', oq: 'search', gs_lcrp: 'EgZjaHJvbWUyCwgAEEUYDBg5GIAEMgkI
▶ ARAAGAwYgAQyCQgCEA…BFGDwyBggGEEUYPDIGCAcQRRg80gEIMjIzOWowajSoAgCwAgA',
sourceid: 'chrome', ie: 'UTF-8'}
```

```
> getQuery(location.href)
```

```
{q: 'search', oq: 'search', gs_lcrp: 'EgZjaHJvbWUyCwgAEEUYDBg5GIAEMgkI
▶ ARAAGAwYgAQyCQgCEA…BFGDwyBggGEEUYPDIGCAcQRRg80gEIMjIzOWowajSoAgCwAgA',
sourceid: 'chrome', ie: 'UTF-8'}
```

```
> str = 'Stephen'
< 'Stephen'
> str.substr(0,3)
< 'Ste'
> str.substr(3)
< 'phen'
```

```
1  // split出来的是 array
2  str = 'string';
3  array = str.split('r');
4  console.log(array);
5  dict = {};
6  dict[array[0]] = array[1];
7  console.log(dict)
8
```

```
问题 9  输出  调试控制台  终端
/usr/local/bin/node ./webAPI资料源码/draft.js
> (2) ['st', 'ing']
> {st: 'ing'}
```

location对象是window对象下的一个属性，时候的时候可以省略window对象

location可以获取或者设置浏览器地址栏的URL

- 使用chrome的控制台查看
- 查MDN MDN
- 成员
  - assign()/reload()/replace()
  - hash/host/hostname/search/href……
-

- URL的组成

```
scheme://host:port/path?query#fragment
scheme:通信协议
  常用的http,ftp,maito等
host:主机
  服务器(计算机)域名系统 (DNS) 主机名或 IP 地址。
port:端口号
  整数，可选，省略时使用方案的默认端口，如http的默认端口为80。
path:路径
  由零或多个'/'符号隔开的字符串，一般用来表示主机上的一个目录或文件地址。
query:查询
  可选，用于给动态网页传递参数，可有多个参数，用'&'符号隔开，每个参数的名和值用'='符号隔开。例如：na
fragment:信息片断
  字符串，锚点.
```

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);
```

BOM的顶级对象window

window是浏览器的顶级对象，当调用window下的属性和方法时，可以
注意：window下一个特殊的属性 window.name

对话框

- alert()
- prompt()
- confirm()

页面加载事件

- onload

```
window.onload = function () {
  // 当页面加载完成执行
  // 当页面完全加载所有内容（包括图像、脚本文件、CSS 文件等）执行
}
```

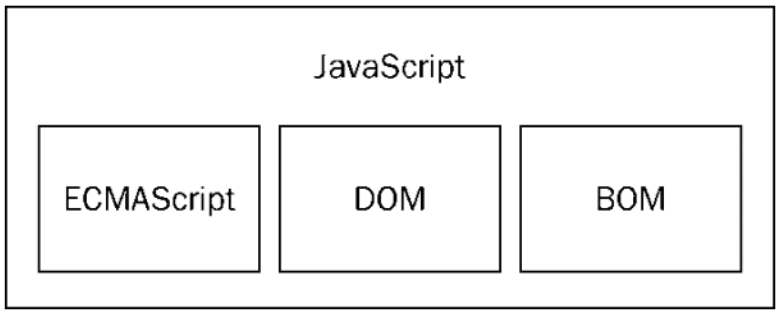
- onunload

```
window.onunload = function () {
  // 当用户退出页面时执行
}
```

API科普

https://developer.mozilla.org/zh-CN/docs/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

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)

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())