# HTML5\_API

一、兼容处理

在不支持HTML5新标签的浏览器里，会将这些新的标签解析成行内元素(inline)对待，所以我们只需要将其转换成块元素(block)即可使用。这个就是在base.css中，为什么会把元素display：block。

但是在IE9版本以下，并不能正常解析这些新标签，但是却可以识别通过document.createElement('tagName')创建的自定义标签，于是我们的解决方案

1、处理方式：

（1）就是将HTML5的新标签全部通过document.createElement('tagName')来创建一遍，创建后**默认是行内元素**，这样IE低版本也能正常解析HTML5新标签了。这个方式麻烦

（2）在实际开发中我们更多采用的是通过检测IE浏览器的版本来加载三方的一个JS库来解决兼容问题（测试在IE下面的兼容性：ieTester软件的使用），**默认是块级元素**

2、条件注释

类似于条件编译， 主要是针对于ie低版本的，也就是只有低版本ie才执行才对。

|  |
| --- |
| <!--[if lt IE 9]>  　　 <script src="js/respond.js"></script>    　　<script src="js/html5shiv.min.js"></script>  <![endif]—> |

respond.js 目的是为了解决 ie低版本的CSS3媒体查询 media query（PC端是一个样式，在手机端又是另外的样式）

条件注释 了解

|  |
| --- |
| **<!--[if !IE]><!--> 除IE外都可识别 <!--<![endif]-->**  <!--[if IE]> 所有的IE可识别 <![endif]-->  <!--[if IE 6]> 仅IE6可识别 <![endif]-->  **<!--[if lte IE 6]> IE6以及IE6以下版本可识别 <![endif]-->**  <!--[if gte IE 6]> IE6以及IE6以上版本可识别 <![endif]-->  <!--[if IE 7]> 仅IE7可识别 <![endif]-->  **<!--[if lt IE 7]> IE7以下版本可识别 <![endif]-->**  <!--[if gt IE 7]> IE7以上版本可识别 <![endif]-->  <!--[if IE 8]> 仅IE8可识别 <![endif]-->  <!--[if IE 9]> 仅IE9可识别 <![endif]--> |

二、网络状态

我们可以通过window.navigator.onLine来检测，用户当前的网络状况，返回一个布尔值， 但是不同浏览器会存在差异。

HTML5 给我们提供了2个事件 online 和 offline

1、online用户网络连接时被调用

2、offline用户网络断开时被调用

他们监听的对象都是window。

**无法检测联网的情况。。？？？怀疑是电脑的问题**

|  |
| --- |
| <script type="text/javascript" src="jquery-1.12.2.js"></script>  <style type="text/css">  p {  width: 200px;  height: 100px;  background-color: black;  color: white;  font-weight: 900;  line-height: 100px;  text-align: center;  display: block;  }  </style>  <body>  <p id="show">hello</p>  <script type="text/javascript">  console.log(window.navigator.onLine);  $(function () {    window.addEventListener('online', function () {  $('#show').text('网络已连接');  });  window.addEventListener('offline', function () {  $('#show').text('网络已断开');  });  });  </script>  </body> |

三、全屏

HTML5规范允许用户自定义网页上任一元素全屏显示。例如看视频的时候，全屏播放

1、element.requestFullScreen() 开启全屏显示

（1）webkit内核浏览器：webkitRequestFullScreen、webkitCancelFullScreen，如chrome浏览器。

（2）Gecko内核浏览器：mozRequestFullScreen、mozCancelFullScreen，如火狐浏览器。

（3）ms微软：msRequestFullscreen，它的screen中的 s 是小写的，跟其他不一样。

1. o 欧朋： oRequestFullScreen。

各个浏览器的全屏效果是有区别的。

2、document.cancelFullScreen() 关闭全屏显示，取消全屏跟元素是没有关系的，跟 document 有关系。

（1）webkitCancelFullScreen()

（2）msExitFullscreen()，IE，微软

由于其兼容性原因，不同浏览器需要添加前缀如：ie9 + 才支持的，即是高版本的浏览器也有兼容性问题，内核不一样需要写不同的前缀

3、document.fullScreen检测当前是否处于全屏，返回 true 或者 false

（1）document.webkitIsFullScreen 返回布尔类型。

（2）document.msFullscreenElement 返回全屏的元素，否则返回null。

4、全屏伪类选择器 也有兼容性问题 需要添加前缀，类似 :hover 的用法

:full-screen 、:-webkit-full-screen {}、:moz-full-screen {}

ie的目前还没有找到

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>HTML5练习</title>  <script type="text/javascript" src="jquery-1.12.2.js"></script>  <style type="text/css">  img {  width: 200px;  }  div:-**webkit**-full-screen {  background-color: deeppink;  }  </style>  </head>  <body>  <div class="container">  <img src="images/pic1.png">  <input type="button" value="全屏" id="fullScreen">  <input type="button" value="取消全屏" id="cancelScreen">  <input type="button" value="检测是否全屏" id="checkScreen">  </div>  <script type="text/javascript">  var fullScreen = document.getElementById('fullScreen');  var box = document.querySelector('.container');  fullScreen.onclick = function () {  if (box.requestFullScreen) {  box.requestFullScreen();  } else if (box.webkitRequestFullScreen) {  box.webkitRequestFullScreen();  } else if (box.msRequestFullscreen) {  box.msRequestFullscreen();  } else if (box.oRequestFullScreen) {  box.oRequestFullScreen();  } else if (box.mozRequestFullScreen) {  box.mozRequestFullScreen();  }  };  var cancelScreen = document.getElementById('cancelScreen');  cancelScreen.onclick = function () {  if (document.webkitCancelFullScreen) {  document.**webkitCancelFullScreen**();  } else if (document.msExitFullscreen) {  document.**msExitFullscreen**();  }  }  var checkScreen = document.getElementById('checkScreen');  checkScreen.onclick = function () {  if (typeof document.**fullScreen** === 'boolean') {  alert(document.fullScreen);  } else if (typeof document.**webkitIsFullScreen** === 'boolean') {  alert(document.webkitIsFullScreen + ' webkit');  } else if (typeof document.**msFullscreenElement** === 'object') {  // 返回全屏的元素，否则返回null  alert(document.msFullscreenElement + ' ms');  }  }  </script>  </body>  </html> |

四、文件读取

可以把上传的文件，内容显示到页面？或者上传完毕图片显示缩略图到页面上......

1、上传我们的文件，借助于 文件域 input type = file

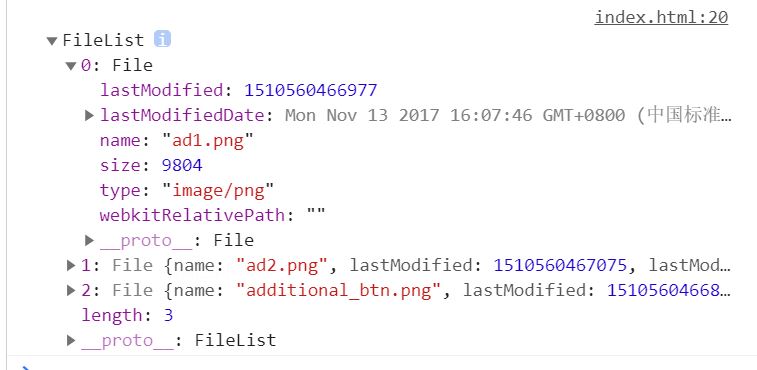
2、选择我们要的文件，通过 fileReader 进行读取

3、把读取到的内容显示到页面中

onchange()，通过获取DOM元素获取里边的Files对象中的FileList，可以是一个文件或很多文件。

（1）files对象

由于HTML5中我们可以通过为表单元素添加multiple属性，因此我们通过<input>上传文件后得到的是一个Files对象（伪数组形式）。



（2）FileReader对象

读取内容的时候要注意编码的问题，例如文字采用了什么编码，否则会出现乱码情况。

HTML5新增内建对象，可以读取本地文件内容。

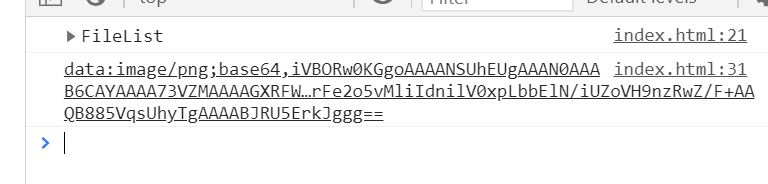
var reader = new FileReader(); 可以实例化一个对象。

|  |
| --- |
| <div class="container">  <input type="file" id="file" multiple />  </div>  <div id="showTxt"></div>  <script type="text/javascript">  var file = document.querySelector('input');  **file.onchange** = function () {  console.dir(file.files);  **var reader = new FileReader();**  // 参数是一个文件对象，可以是其他，见MDN文档  **reader.readAsText(file.files[0]);**  // 当这个文件读取完毕之后，就会把内容存放到 result 中  // onload 表示 reader 已经读取完成，而且读取到的内容，存放到 result 里边了  **reader.onload** = function () {  var show = document.querySelector('#showTxt');  show.innerHTML = reader.result;  }  }  </script> |

（3）生成缩略图形式，readAsDataURL()

采用 readAsDataURL() ，读取的是图片的base64的编码，直接给 img 的 src 属性就可以。

同时 result 属性将包含一个data:URL格式的字符串（base64编码）以表示所读取文件的内容。



|  |
| --- |
| <div class="container">  <input type="file" id="file" multiple />  </div>  <img src="" alt="" id="show" width="200">  <script type="text/javascript">  var file = document.querySelector('input');  file.onchange = function () {  console.dir(file.files);  var reader = new FileReader();  // 参数是一个文件对象，可以是其他，见MDN文档  reader.**readAsDataURL**(file.files[0]);  // 当这个文件读取完毕之后，就会把内容存放到 result 中  // onload 表示 reader 已经读取完成，而且读取到的内容，存放到 result 里边了  reader.onload = function () {  var show = document.querySelector('#show');  show.src = reader.result;  console.log(reader.result);  }  }  </script> |

Blob 对象表示一个不可变、原始数据的类文件对象。它的数据可以按文本或二进制的格式进行读取，也可以转换成 ReadableStream 来用于数据操作。

地图名片的使用

五、地理定位

在HTML规范中，增加了获取用户地理信息的API，这样使得我们可以基于用户位置开发互联网应用，即基于位置服务 (Location Base Service)。

HTML用的是谷歌的地图，用的是谷歌的定位，在中国浏览器端定位不了，可以使用百度地图api

1、API详解

（1）获取当前地理信息

navigator.geolocation.getCurrentPosition(successCallback, errorCallback)

（2）重复获取当前地理信息

navigator. geolocation.watchPosition(successCallback, errorCallback)

当成功获取地理信息后，会调用succssCallback，并返回一个包含位置信息的对象position。

position.coords.latitude 纬度

position.coords.longitude经度

position.coords.accuracy精度

position.coords.altitude海拔高度

当获取地理信息失败后，会调用errorCallback，并返回错误信息error

http://www.w3school.com.cn/html5/html\_5\_geolocation.asp

2、应用

在现实开发中，通过调用第三方API（如百度地图）来实现地理定位信息，这些API都是基于用户当前位置的，并将用位置位置（经/纬度）当做参数传递，就可以实现相应的功能。

3、百度地图api

http://lbsyun.baidu.com/ 百度地图api

在使用的时候，要把百度地图的api地址补充完整

<script type="text/javascript" src="**http:**//api.map.baidu.com/api?v=3.0&ak=53t3yYms0TpLQCmUzXH8a26dRG0DCnQj"></script>

六、拖拽

在 HTML5 中，拖放是标准的一部分，任何元素都能够拖放。

1、拖拽和释放

拖拽：Drag。

释放：Drop。

拖拽指的是鼠标点击源对象后一直移动对象不松手，一但松手即释放了。

2、设置元素为可拖放

draggable 属性：就是标签元素要设置draggable=true，否则不会有效果。

注意：链接和图片默认是可拖动的，不需要 draggable 属性。

（1）我们有很多盒子，哪个需要拖放，就给这个盒子设置成“有拖放的权限”。

（2）找到源元素，设置拖拽效果。

（3）找到目标元素，设置释放效果。

3、拖拽API的相关事件

（1）被拖动的源对象可以触发的事件：

ondragstart：源对象开始被拖动。执行一次。

ondrag：源对象被拖动过程中(鼠标可能在移动也可能未移动)。会一直不断的执行，被触发。

ondragend：源对象被拖动结束。执行一次。

|  |
| --- |
| <style type="text/css">  div {  width: 200px;  height: 200px;  margin: 10px;  float: left;  border: 2px solid pink;  }  div:first-child {  border: 2px solid #000;  }  div:last-child {  border: 2px solid yellow;  }  p {  text-align: center;  background-color: lightblue;  }  </style> |
| <body>  <div id="box1">  <p id="p1" draggable="true">hello</p>  </div>  <div id="box2"></div>  <div id="box3"></div>  <script type="text/javascript">  var p1 = document.querySelector('#p1');  p1.ondragstart = function () {  console.log('ondragstart');  }  p1.ondrag = function () {  console.log('ondrag');  }  p1.ondragend = function () {  console.log('ondragend');  }  </script>  </body> |
| 1 |

（2）拖动源对象可以进入到上方的目标对象可以触发的事件：

ondragenter：目标对象被源对象拖动着进入。是鼠标跟着进入的时候，才会触发，如果只是盒子的一部分进入是不会触发的。

ondragover：目标对象被源对象拖动着悬停在上方。会一直不断的执行，被触发。如果想要触发 ondrop 事件，则需要在 over 里面阻止默认行为。

ondragleave：源对象拖动着离开了目标对象。

ondrop：源对象拖动着在目标对象上方释放/松手。浏览器默认是阻止的？？？

当源对象在目标对象上释放的时候，就可以把“源对象”放入目标对象里边，追加就行了：target.appendChild(origin)。这样也不用去删除原来的位置。

|  |
| --- |
| var box2 = document.querySelector('#box2');  box2.**ondragenter** = function () {  console.log('ondragenter 进入目标');  }  box2.**ondragover** = function (e) {  console.log('ondragover 目标之上');  e.preventDefault();  // return false;  }  box2.**ondragleave** = function () {  console.log('ondragleave 离开目标');  }  box2.**ondrop** = function () {  console.log('ondrop 放在目标');  }  释放的不要写成 drag 了 |

拖拽API总共就是7个函数！都支持事件冒泡，可以使用事件委托，直接给整个文档document去处理，就能拖来拖去的。

4、DataTransfer

DataTransfer 对象用于保存拖动并放下（drag and drop）过程中的数据。它可以保存一项或多项数据、一种或者多种数据类型。

（1）获取方式：这个对象可以从所有拖动事件 drag events 的 dataTransfer 属性上获取

它是在存在于拖动事件中的，就可以在event事件中找到。可以不用全局变量来接收触发事件的事件源对象。

语法

（2）void dataTransfer.setData(format, data);

参数format

表示要添加到 drag object的拖动数据的类型。

data

表示要添加到 drag object的数据。

返回值

void

（3）DOMString dataTransfer.getData(format);

参数format

数据类型

返回值

返回一个给定类型的DOMString格式的数据。如果没有操作数据或者没有指定操作数据的类型，都会返回一个空字符串。

数据类型： text/html、URL

|  |
| --- |
| // var origin = null;  document.ondragstart = function (e) {  // origin = e.target;  e.dataTransfer.**setData**('text/html', e.target.id);  }  document.ondragover = function () {  return false;  }  document.ondrop = function (e) {  // e.target.appendChild(origin);  e.target.appendChild(document.getElementById(e.dataTransfer.**getData**('text/html')));  } |

七、Web存储

随着互联网的快速发展，基于网页的应用越来越普遍，同时也变的越来越复杂，为了满足各种各样的需求，会经常性在本地存储大量的数据，HTML5规范提出了相关解决方案。

cookie的容量非常小，4kb，每次使用都会发送请求

1、特性

（1）设置、读取方便、页面刷新不丢失数据。

（2）容量较大，sessionStorage约5M、localStorage约20M。

（3）只能存储字符串，可以将对象JSON.stringify() 编码后存储。

2、window.sessionStorage

（1）生命周期为关闭浏览器窗口。关闭页面数据就被删除了，页面刷新是不会丢失的。

（2）在同一个窗口(页面)下数据可以共享。

|  |
| --- |
| <body>  <input type="text" id="txt"><br />  <input type="button" id="add" value="添加数据">  <input type="button" id="get" value="获取数据">  <input type="button" id="remove" value="删除所有数据">  <script type="text/javascript">  var txt = document.querySelector("#txt");  var add = document.querySelector("#add");  var get = document.querySelector("#get");  var removeAll = document.querySelector("#removeAll");  window.sessionStorage.setItem('password', '123');  window.sessionStorage.setItem('admin', 'hello');  add.onclick = function () {  window.sessionStorage.setItem('username', txt.value);  }  get.onclick = function () {  var str = window.sessionStorage.getItem('username');  console.log(str);  }  remove.onclick = function () {  window.sessionStorage.**removeItem**('username');  // window.sessionStorage.**clear**();  }  </script>  </body>  1 |

3、window.localStorage

（1）永久生效，除非手动删除，关闭页面也会存在。

（2）可以多窗口（页面）共享（同一浏览器可以共享，同一个作用域下）

|  |
| --- |
| var txt = document.querySelector("#txt");  var add = document.querySelector("#add");  var get = document.querySelector("#get");  var removeAll = document.querySelector("#removeAll");  add.onclick = function () {  window.localStorage.setItem('username', txt.value);  }  get.onclick = function () {  var str = window.localStorage.getItem('username');  console.log(str);  }  remove.onclick = function () {  window.localStorage.removeItem('username');  // window.localStorage.clear();  } |
| 1  2 |

4、方法详解

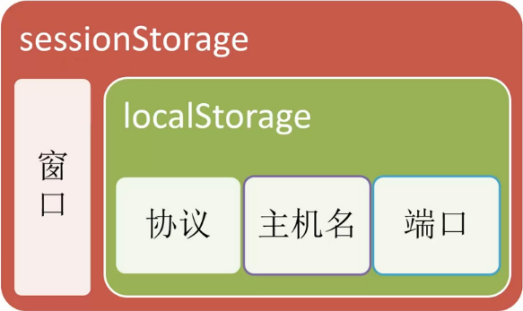
（1）setItem(key, value)：设置存储内容

（2）getItem(key)：读取存储内容

（3）removeItem(key)：删除键值为key的存储内容

（4）clear()：清空所有存储内容，这个清除方式尽量不要使用，一不小心，删除所有，它造成的后果是比较严重的，特别是localStroage，它的数据是整个浏览器公用的

5、作用域



这里的作用域指的是：如何隔离开不同页面之间的localStorage（总不能在百度的页面上能读到腾讯的localStorage吧，哈哈哈）。

localStorage只要在相同的协议、相同的主机名、相同的端口下，就能读取/修改到同一份localStorage数据。

sessionStorage比localStorage更严苛一点，除了协议、主机名、端口外，还要求在同一窗口（也就是浏览器的标签页）下。

6、生存期

（1）localStorage理论上来说是永久有效的，即不主动清空的话就不会消失，即使保存的数据超出了浏览器所规定的大小，也不会把旧数据清空而只会报错。但需要注意的是，在移动设备上的浏览器或各Native App用到的WebView里，localStorage都是不可靠的，可能会因为各种原因（比如说退出App、网络切换、内存不足等原因）被清空。

（2）sessionStorage的生存期顾名思义，类似于session，只要关闭浏览器（也包括浏览器的标签页），就会被清空。由于sessionStorage的生存期太短，因此应用场景很有限，但从另一方面来看，不容易出现异常情况，比较可靠。

八、应用缓存（没有服务器，暂时没有进行验证，只是知道有这东西）

HTML5中我们可以轻松的构建一个离线（无网络状态）应用，只需要创建一个cache manifest文件。

1、优势

（1）可配置需要缓存的资源，可以只缓存页面中的一部分

（2）网络无连接应用仍可用

（3）本地读取缓存资源，提升访问速度，增强用户体验

（4）减少请求，缓解服务器负担

2、缓存清单

一个普通文本文件，其中列出了浏览器应缓存以供离线访问的资源，推荐使用 .appcache 为后缀名。

1. 创建了一个名为demo.appcache的文件，

（2）然后在需要应用缓存在页面的根元素(html)添加属性manifest="demo.appcache"，路径要保证正确。引入这个缓存清单。

3、manifest文件格式\*

（1）顶行写CACHE MANIFEST。

（2）CACHE: 换行，指定我们需要缓存的静态资源，如.css、image、js等。

（3）NETWORK: 换行，指定需要在线访问的资源，可使用通配符。

（4）FALLBACK: 换行，当被缓存的文件找不到时的备用资源。

（5）\* 表示所有。

4 其它

（1）CACHE: 可以省略，这种情况下将需要缓存的资源写在CACHE MANIFEST

（2）可以指定多个CACHE: NETWORK: FALLBACK:，无顺序限制

（3）#表示注释，只有当demo.appcache文件内容发生改变时或者手动清除缓存后，才会重新缓存。

（4）chrome 可以通过chrome://appcache-internals/工具和离线（offline）模式来调试管理应用缓存

1. video案例

在全屏之后，自定义的控件没法显示问题。目前没找到相关的资料，有的是全屏化它外一层的盒子。

这些方法、属性和事件允许使用 javascript 来操作 <audio> 和 <video> 元素，这些 api 是原生的 js 操作，如果使用其他库，如jQuery，需要进行转换之后，再进行调用。

时间的计算还要再思考一下

它获取位置，使用的是e.offsetX

ontimeupdate，只要修改了时间，它也会去触发事件

translate 相当于相对定位，从自己现在的位置开始移动距离

在动画中采用了translate是因为它可以回到自己的位置，我们很难去为每个圆都去写一个动画，它们的left值都不一样。

中心位置放大，缩小原理：

可以采用父级perspective 跟 translateZ来实现，再加上过渡transition效果，如果采用设置宽高来控制缩放，很难处一直处于中心点

背景旋转的话，可以看成一个整体，来旋转整个背景即可。

transform: scale(.3)缩小到原来的0.3，即30%

jQuery的animate是不支持transform等

jQuery 里面通过css 和 transition 搭配也能基本实现类似于 animate 的效果

有时要多刷新，不然重新打开页面

十、Canvas

canvas 元素用于在网页上绘制图形。

1、什么是 Canvas？

HTML5 的 canvas 元素使用 **JavaScript** 在网页上绘制图像。

画布是一个矩形区域，您可以控制其每一像素。

canvas 拥有多种绘制路径、矩形、圆形、字符以及添加图像的方法。

2、创建 Canvas 元素

向 HTML5 页面添加 canvas 元素。

规定元素的 id、宽度和高度：

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Document</title>  <style type="text/css">    </style>  </head>  <body>  <canvas id="canvas" width="200" height="200"></canvas>  <script type="text/javascript">  var canvas = document.getElementById('canvas');  var cxt = canvas.getContext('2d');  cxt.fillStyle = '#FF0000';  cxt.fillRect(100,0,100,100);  </script>  </body>  </html> |

3、通过 JavaScript 来绘制

canvas 元素本身是**没有绘图能力**的。所有的绘制工作必须在 **JavaScript** 内部完成，如果超出画布的区域那就看不见了。

1. JavaScript 使用 id 来寻找 canvas 元素
2. 然后，通过getContext创建 context 对象：

十一、cookie相关知识(不属于H5新增的)

**以www这种访问方式才有cookie存在，如果是file的不会有**

Cookie 是在计算机上存储在小的文本文件中的数据。

当 web 服务器向浏览器发送网页后，连接被关闭，服务器会忘记用户的一切。

Cookie 是为了解决“如何记住用户信息”而发明的：

•当用户访问网页时，他的名字可以存储在 cookie 中。

•下次用户访问该页面时，cookie 会“记住”他的名字。

当浏览器从服务器请求一个网页时，将属于该页的 cookie 添加到该请求中。这样服务器就获得了必要的数据来“记住”用户的信息。



如果浏览器已关闭本地 cookie 支持，则以下实例均无法工作。

默认情况下，在浏览器关闭时会删除 cookie，如果浏览器一直打开，它是不会被删除的，即使页面被关闭。

1、通过 JavaScript，可以这样创建 cookie：

document.cookie = "username=Bill Gates";

2、可以通过expires参数添加有效日期（UTC 时间）。默认情况下，在浏览器关闭时会删除

注意：UTC时间跟我们时间是相差了8个小时的，现在是变成GMT结尾了

document.cookie = 'username=lxf; expires=Fri, 18 Jun 2021 03:43:00 GMT; path=/;';

3、通过 path 参数，您可以告诉浏览器 cookie 属于什么路径。默认情况下，cookie 属于当前页。

document.cookie = "username=Bill Gates; expires=Sun, 31 Dec 2017 12:00:00 UTC; path=/";

4、修改 cookie

你可以像你创建 cookie 一样改变它，就是重新给他赋值

document.cookie = "username=Steve Jobs; expires=Sun, 31 Dec 2017 12:00:00 UTC; path=/";

旧 cookie 被覆盖。

5、通过 JavaScript 删除 cookie

删除 cookie 非常简单。删除 cookie 时不必指定 cookie 值：

直接把 **expires 参数设置为过去的日期即可**：

document.cookie = "username=; **expires**=Thu, 01 Jan 1970 00:00:00 UTC; path=/;";

您应该定义 cookie 路径以确保删除正确的 cookie。

如果你不指定路径，一些浏览器不会让你删除 cookie。

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Document</title>  </head>  <body>  <input type="text" name="" id="txt">  <input type="button" value="提交cookie名字" id="btn">  <script type="text/javascript">  var btn = document.getElementById('btn');  var txt = document.getElementById('txt');  btn.onclick = function () {  var text = txt.value;  // 设置cookie的时间只有一分钟  var date = new Date().getTime() + 60 \* 1000;  date = new Date(date);  var str = `username**=**${text}; expires**=**${date.toUTCString()}; path=/;`;  console.log(str);  document.cookie = str;  }  </script>  </body>  </html>  1 |

**6、浏览器允许每个域名所包含的cookie数：**

　　Microsoft指出InternetExplorer8增加cookie限制为每个域名50个，但IE7似乎也允许每个域名50个cookie。

　　Firefox每个域名cookie限制为50个。

　　Opera每个域名cookie限制为30个。

**Safari/WebKit貌似没有cookie限制。但是如果cookie很多，则会使header大小超过[服务器](http://product.it168.com/files/0402search.shtml)的处理的限制，会导致错误发生。**

**7、不同浏览器间cookie总大小也不同：大约是4k字节**

　　Firefox和Safari允许cookie多达4097个字节，包括名（name）、值（value）和等号。

　　Opera允许cookie多达4096个字节，包括：名（name）、值（value）和等号。

　　Internet Explorer允许cookie多达4095个字节，包括：名（name）、值（value）和等号。

      注：多字节字符计算为两个字节。在所有浏览器中，任何cookie大小超过限制都被忽略，且永远不会被设置。

1. \*封装设置以及获取cookie的函数

在解析需要注意name前边的空格，它在比较的时候会影响，需要对空格进行处理



|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Document</title>  </head>  <body>      <div class="box">          <label>cookie名字：<input type="text" id="name"></label><br />          <label>cookie值：<input type="text" id="value"></label><br />          <input type="button" value="提交" id="btn">          <input type="button" value="删除" id="btn1">      </div>      <canvas id="canvas" width="200" height="200"></canvas>      <script type="text/javascript">          var btn = document.getElementById('btn');          btn.onclick = function () {              var name = document.getElementById('name').value;              var value = document.getElementById('value').value;              setCookie(name, value, 1);              getCookie(name);          }          btn1.onclick = function () {              var name = document.getElementById('name').value;              delCookie(name);          }  **/\***  **name：cookie名字**  **value：cookie值**  **day: 距离当前时间点，多少天后过期**  **一些参数错误的还没有去判断**  **\*/**  **function setCookie(name, value, day) {**  **var current = new Date();**  **var expiresDay = new Date(current.getTime() + day \* 24 \* 60 \* 60 \* 1000);**  **var str = `${name}=${value}; expires=${expiresDay.toUTCString()};`;**  **document.cookie = str;**  **}**  **/\***  **getCookie会返回结果,没有找到就返回 -1**  **\*/**  **function getCookie(name) {**  **var value = document.cookie;**  **var result = value.split(/;|=/);**  **for (var i = 0, len = result.length; i < len; i+=2) {**  **// cookie的第二个解析后前边会带有空格,trim会返回新的字符串**  **if (result[i].trim() === name) {**  **// 它的值就跟在键的后边**  **console.log(result[i+1]);**  **return result[i+1];**  **}**  **}**  **return -1;**  **}**  **function delCookie(name) {**  **// 先判断是否有name这个cookie**  **var value = getCookie(name);**  **if (value !== -1) {**  **var current = new Date();**  **// 设置成过去的时间就删除了**  **var expiresDay = new Date(current.getTime() - 1);**  **var str = `${name}=${value}; expires=${expiresDay.toUTCString()};`;**  **document.cookie = str;**  **}**  **}**      </script>  </body>  </html> |