



- 一、HTML结构
  - 1. DOCTYPE
    - 1-1. DOCTYPE是什么?
    - 1-2. DOCTYPE分类?
  - 2. HTML标签
    - 2-1. html标签
    - 2-2. head标签
    - 2-3. 行内、块级等元素分类
    - 2-4. 几种标签作用
      - 2-4-1. href 和 src的区别?
      - 2-4-2. 为什么我们要弃用table标签?
      - 2-4-3. form表单元素的作用?
      - 2-4-4. HTML5 的 form 如何关闭自动补全功能?
      - 2-4-5. img的alt属性和title属性有何区别?
      - 2-4-6. 说说title属性?
      - 2-4-7. label的作用是什么? 怎么使用?
      - 2-4-8. iframe是什么? 优缺点?
  - 3. 其他
    - 3-1. HTML语义化
      - 3-1-1. 语义化的意义?
      - 3-1-2. 语义化举例?
    - 3-2. Web Quality是什么?
    - 3-3. document.write 和 innerHTML 的区别
    - 3-4. innerText属性和innerHTML属性的区别
- 二、HTML5
  - 1. HTML5特性?
    - 1-1. 增删元素?
    - 1-2. 区分与兼容性?
  - 2. 新的API?
    - 2-1. canvas和svg区别?
    - 2-2. 离线储存 applicationCache 工作原理?
      - 2-2-1. 浏览器是怎么对 HTML5 的离线储存资源进行管理和加载的呢?



- 2-2-2. applicationCache的优点?
- 2-3. ★本地存储Cookie、LocalStorage、SessionStorage的区别?
- 2-4. WebSocket的实现和应用
  - 2-4-1. WebSocket和HTTP的区别?
  - 2-4-2. 什么是WebSocket?
  - 2-4-3. WebSocket具体有什么优点?
  - 2-4-4. WebSocket如何兼容低版本浏览器?
- 2-5. WebWorker的实现和应用
  - 2-5-1. WebWorker是什么, 有什么优缺点?
  - 2-5-2. WebWorker的创建方法?
  - 2-5-3. WebWorker使用注意事项?
- 2-6. SharedWorker是什么?
  - **共享worker(SharedWorker)**, 这种是可以多个标签页、iframe共同使用的, 可以实现标签页之间的通信。SharedWorker可以被多个window共同使用, 但必须保证这些标签页都是同源的(相同的协议, 主机和端口号)
  - 2-6-1. SharedWorker的创建方法?
- 2-7. drag API包含哪些?
- 2-8. Geolocation地图定位
- 2-9. 页面可见性 (Page Visibility API) 可以有哪些用途?
- 3. 应用?
  - 3-1. 如何实现浏览器内多个标签页之间的通信?
  - 3-2. 如何实现实时通信?

# 一、HTML结构

```
<!DOCTYPE html>
<html lang="en">
<head>
  <!--网页页面字符集-->
  <meta charset="UTF-8">

  <!--针对移动设备,网站显示宽度等于设备屏幕显示宽度,内容缩放比例为1:1-->
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <!--让IE使用最新的渲染模式-->
  <meta http-equiv="X-UA-Compatible" content="ie=edge">

  <!--将下面的 <meta> 标签加入到页面中,可以让部分国产浏览器默认采用高速模式渲染
```

☰

```
<meta name="renderer" content="webkit">
<!-- 上述3个meta标签*必须*放在最前面，任何其他内容都*必须*跟随其后! -->

<title>xx页面</title>

<!-- css样式文件放在head标签尾部，body的前面：边解析css边解析dom，解析完一起渲染 -->
<link href="css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
  <div id="container">default</div>
  <p>test</p>

  <!-- js脚本文件放在body标签尾部，防止阻塞渲染 -->
  <script src="index.js"></script>
  <!-- 这里src引用的是本地，线上建议使用CDN引用 -->
</body>
</html>
```

☰

## 1. DOCTYPE

### 1-1. DOCTYPE是什么？

- DOCTYPE是用来声明**文档类型**和DTD（Document Type Definition）规范的。浏览器据此来选择用什么引擎去**解析/渲染**它。
- `<!DOCTYPE html>` 声明位于HTML文档中的第一行，不是一个HTML标签，处于 `html` 标签之前。DOCTYPE不存在或格式不正确会导致文档以**兼容模式**呈现。

### 1-2. DOCTYPE分类？

- **HTML5的模式** `<!DOCTYPE html>`，不像HTML4基于SGML（标准通用标记语言），所以不用指定DTD；
- **HTML4的严格/标准模式**，会声明一个**strict.dtd**规范，表明该DTD包含所有HTML元素和属性，但是**不包括**展示性和弃用的元素，比如font。该模式以该浏览器支持的最高标准运行；
- **HTML4的传统/兼容模式**，会声明一个**loose.dtd**规范，表明该DTD包含所有HTML元素和属性，**包括**展示性和弃用的元素。页面以宽松的向后兼容的方式显示，模拟老式浏览器的行为以防止站点无法工作。

## 2. HTML标签

### 2-1. html标签

`<html lang="en">`，**语言**，必须定义并且包含在 `html` 标签中

## 2. head标签



- **head**标签的**meta**元素大概分为三类，分别是：

- 描述**网页基本信息**的
- 指向渲染网页需要**其他文件链接**的
- 各大厂商根据自己需要**定制**的

- **1. 网页基本信息**

- **文档标题**（浏览器标签中显示的文本）：`<title>`
- **字符编码**：如果页面出现乱码，一般是编码格式不对 `<meta charset="utf-8">`
- **视窗设置**：  
`<meta name="viewport" content="width=device-width, initial-scale=1.0">`
- 搜索引擎优化相关内容：`<meta property="og:description" content="前端故事">`
- **兼容IE**设置：`<meta http-equiv="X-UA-Compatible" content="ie=edge">`

- **2. 其他文件链接**

- CSS 文件：

```
<link rel="stylesheet" href="style.css" type="text/css" media="all" />
```

- JavaScript 文件：

```
<script type="text/javascript" src="integrator.js"></script>
```

但是为了让页面的样子更早的让用户看到，一般把JS文件放到body的底部

- **3. 厂商定制** 同样分享页面到QQ的聊天窗口，有些页面直接就是一个链接，但是有些页面有标题，图片，还有文字介绍。为什么区别这么明显呢？其实就是看有没有设置下面这三个内容

```
<meta itemprop="name" content="这是分享的标题"/>
<meta itemprop="image" content="http://imgcache.qq.com/qshow/ac/v4/global/1
<meta name="description" itemprop="description" content="这是要分享的内容" />
```

### 2-3. 行内、块级等元素分类

定义：CSS 规范规定，每个元素都有 **display**属性，确定该元素的类型，每个元素都有默认的 display 值，如 div 的 display 默认值为“**block**”，则为“**块级**”元素；span 默认 display 属性值为“**inline**”，是“**行内**”元素。

- 行内元素(display:inline)
  - **a b span img input select strong**
- 块级元素(display:block)
  - **div ul ol li dl dt dd h1 h2 h3 h4...p**
- 空(void)元素



- **br hr img input link meta**



- 自闭和元素

- **input img br hr meta link**

## 2-4. 几种标签作用

### 2-4-1. href 和 src的区别？

- href 表示超文本引用（hypertext reference），在 **link**和**a** 等元素上使用。
  - href 的内容，是与该页面有关联，是引用。
  - `<link href="common.css" rel="stylesheet"/>` 浏览器会识别该文档为css文件，就会**并行下载资源并且不会停止对当前文档的处理**。这也是为什么建议使用link方式来加载css，而不是使用@import方式。
- src 表示来源地址，在 **img**、**script**、**iframe** 等元素上。
  - src 的内容，是页面必不可少的一部分，是引入。
  - `<script src = "js.js"></script>` 当浏览器解析到该元素时，会**暂停其他资源的下载和处理**，直到将该资源**加载、编译、执行完毕**，图片和框架等元素也如此，类似于将所指向资源嵌入当前标签内；这也是为什么将js脚本放在底部而不是头部。

### 2-4-2. 为什么我们要弃用table标签？

- 浏览器从服务器加载代码时，本应是加载一行执行一行，但是table标签是里面的东西**全都下载完之后才会显示出来**，内容很多就会导致很长时间一直加载不出来。

### 2-4-3. form表单元素的作用？

- **直接提交**表单
- 使用**submit / reset**按钮
- 便于浏览器**保存**表单
- 第三方库可以**整体取值**
- 第三方库可以进行**表单验证**

### 2-4-4. HTML5 的 form 如何关闭自动补全功能？

- 给不想要提示的 form 或某个 input 设置为 `autocomplete=off` 。

### 2-4-5. img的alt属性和title属性有何区别？

- ``
- alt属性是在图片未正常输出时显示的文字；
- title属性为图片/链接添加描述性文字，鼠标hover上去出现提示性信息。

## 4-6. 说说title属性?



- title 属性可以用在除了 base, basefont, head, html, meta, param, script 和 title 之外的**所有标签**。
- title 属性的功能是提示。额外的说明信息和非本质的信息请使用 title 属性。title 属性值**可以比 alt 属性值设置的更长**。
- title 属性有一个很好的用途，即为**链接**添加描述性文字，特别是当链接本身并不是十分清楚的表达了链接的目的。

## 2-4-7. label的作用是什么? 怎么使用?

- **label 标签**来定义表单控制间的关系，当用户选择该标签时，浏览器会自动将焦点转到和标签相关的表单控件上。
- ```
<label for="Name">Number:</label> <input type="text" name="Name" id="Name"/>
<label>Date:<input type="text" name="B"/></label>
```
- 两种方式是前后/包住input; 前后需要for属性指向input的name属性; 包住不需要。

## 2-4-8. iframe是什么? 优缺点?

- **定义:** iframe元素会创建包含另一个文档的内联框架
- **提示:** 可以将提示文字放在之间，来提示某些**不支持iframe**的浏览器
- **优点:**
  1. iframe能够把嵌入的网页原封不动的**展现**出来
  2. 如果有**多个网页引用**iframe，那么你只需要修改iframe的内容，就可以实现调用的每一个页面内容的更改，方便快捷
  3. 网页如果**为了统一风格**，头部和版本都是一样的，就可以写成一个页面，用iframe来嵌套，可以**增加代码的可重用**
  4. 如果遇到**加载缓慢**的第三方内容如图标和广告，这些问题可以由iframe来解决
- **缺点:**
  1. iframe会**阻塞主页面的onload事件**
  2. iframe和主页面**共享连接池**，而**浏览器对相同域的连接有限制**，所以会影响页面的**并行加载**，会产生很多页面，不容易管理
  3. iframe框架结构有时会让人感到迷惑，如果框架个数多的话，可能会出现**多个上下、左右滚动条**，会分散访问者的注意力，**用户体验度差**
  4. 代码复杂，无法被一些搜索引擎索引到，这一点很关键，现在的搜索引擎爬虫还不能很好的处理iframe中的内容，所以使用iframe会**不利于搜索引擎优化 (SEO)**



5. 很多的移动设备无法完全显示框架，**移动设备兼容性差**



6. iframe框架页面会**增加服务器的http请求**，对于大型网站是不可取的

- 如果需要使用iframe，最好是通过javascript**动态给iframe添加src属性值**，这样可以避开以上一些问题。

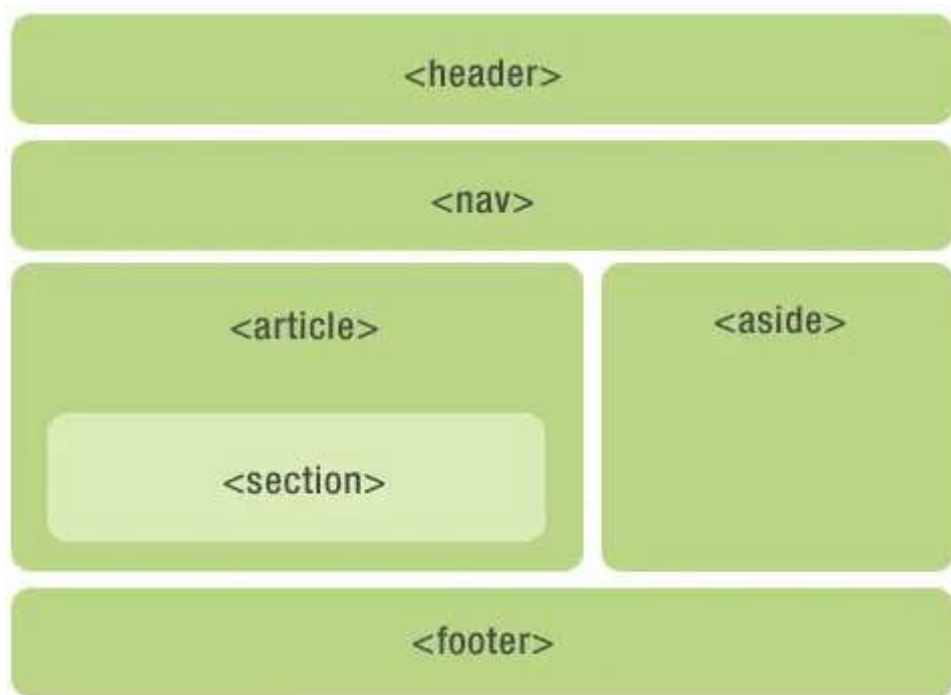
### 3. 其他

#### 3-1. HTML语义化

##### 3-1-1. 语义化的意义？

HTML5语义化标签是指**正确的标签包含正确的内容**

- **代码结构**：html 语义化让页面的内容**结构化**，结构更清晰，便于对浏览器、搜索引擎解析；即使在没有样式 CSS 情况下也以一种文档格式显示，并且是**容易阅读的**
- **有利于SEO**：搜索引擎的爬虫也依赖于 HTML 标记来确定上下文和各个关键字的权重，**利于SEO**（搜索引擎优化）
- **提升用户体验**：使阅读源代码的人对网站更容易将**网站分块**，便于**阅读、维护、理解**
- 比如**nav**表示导航条，类似的还有article、header、footer等等标签。



##### 3-1-2. 语义化举例？

- **em** 标签是强调；  
**i** 标签是斜体，无语义
- **strong** 标签是重点强调； **b** 标签是粗体，无语义
- **h1** 则表示层次明确的标题，对页面信息的抓取也有很大的影响



**title** 没有明确意义只表示是个标题



- 我是code标签，~我是del标签~，我是ins标签【语义化】

### 3-2. Web Quality是什么？

Web品质，分为几个方面：

- **HTML标签元素**，遵循语义化，比如标题的嵌套，还有header、footer、nav等等
- **CSS样式表**，注意背景颜色、字体的一致性，以及可读性
- 细节，排版文字的**行距**，**居中**，避免过于花哨
- **WAI无障碍**，能够被残障人士使用的网站，字体大小的调整，合理利用img的alt属性
- **国际化**，统一**字符集**charset: UTF-8国际标准，国际**日期**格式

### 3-3. document.write 和 innerHTML 的区别

- document.write 只能重绘整个页面
- innerHTML 可以重绘页面的一部分

### 3-4. innerText属性和innerHTML属性的区别

前者会过滤掉html标签获取文本，后者不会过滤。

- innerText属性

```
document.getElementById('box').innerText; //获取文本内容(如有html 直接过滤掉)
document.getElementById('box').innerText = '<div>Mr.Lee</div>'; //设置文本(如有html 直接过滤掉)
```

- innerHTML属性

```
document.getElementById('box').innerHTML; //获取文本(不过滤HTML)
document.getElementById('box').innerHTML = '<b>123</b>'; //可解析成HTML
```

## 二、HTML5

### 1. HTML5特性？

#### 1-1. 增删元素？

- HTML5 现在已经不是 SGML（标准通用标记语言）的子集，主要是关于**图像**，**位置**，**存储**，**多任务**等。
- 增加的元素？





- 语义化元素，比如 **header**、**footer**、**nav**、**aside**、**main**、**article**、**section**
- 内容元素，比如mark高亮、progress进度条
- 表单增强，**color**、**range**、**calendar**、**date**、**time**、**email**、**url**、**search**
- 新的API
  - 图形 (canvas)
  - 音视频 (audio, video)
  - 离线存储 (applicationCache)
  - 本地存储 (localStorage, sessionStorage, indexedDB)
  - 实时通信 (websocket)
  - 设备能力 (Geolocation地图定位, 手机摇一摇)
  - 页面可见性 (Page Visibility)
- 移除的元素？
  - 纯表现的元素：basefont, **big**, center, **font**, s, strike, tt, u;
  - 对可用性产生负面影响的元素：**frame**, **frameset**, noframes;

## 1-2. 区分与兼容性？


- 如何**区分** HTML5: **DOCTYPE 声明** \ **新增的结构元素** \ **功能元素**
- IE8/IE7/IE6 支持通过 **document.createElement**方法产生标签，可以利用这一特性让这些浏览器支持 HTML5 新标签
- 浏览器支持新标签后，还需要添加标签默认的风格
- 当然也可以直接使用**成熟的框架**、比如 **html5shim**;

```
<!--[if lt IE 9]>
  <script>
    src = 'http://html5shim.googlecode.com/svn/trunk/html5.js'
  </script>
<![endif]-->
```

## 2. 新的API？

### 2-1. canvas和svg区别？

- **历史：**
  - **canvas**是html5**提供**的新元素\*\*，而svg存在的历史要比canvas久远，已经有\*\*十几年了。
  - **svg并不是html5**专有的标签，最初svg是用**xml技术**（超文本扩展语言，可以自定义标签或属性）描述二维图形的语言。在H5中看似canvas与svg很像，但是，他们有巨大的差别。
- **应用：**

- 首先，从它们的功能上来讲，**canvas可以看做是一个画布**，其绘制出来的图形为**标量图**，此，可以在canvas中引入jpg或png这类格式的图片，在实际开发中，大型的**网络游戏**都是用canvas画布做出来的，并且canvas的技术现在已经相当的成熟。另外，我们喜欢用canvas来做一些**统计图表**，如柱状图曲线图或饼状图等。
  - 而**svg**，所绘制的图形为**矢量图**，所以其用法上受到了限制。因为只能绘制矢量图，所以svg中不能引入普通的图片，因为矢量图的不会失真的效果，在项目中我们会用来**做小图标**。但是由于其本质为矢量图，可以**被无限放大而不会失真**，这很适合被用来**做地图**，而**百度地图**就是用svg技术做出来的。
- **引擎抓取和事件绑定：**
  - 另外从技术发面来讲canvas里面绘制的图形**不能被引擎抓取**，如我们要让canvas里面的一个图片跟随鼠标事件：`canvas.onmouseover=function(){}` 。
  - 而svg里面的图形**可以被引擎抓取，支持事件的绑定**。另外canvas中我们绘制图形通常是通过javascript来实现，svg更多的是通过标签来来实现，如在svg中绘制正矩形形就要用 `<rect>` ，这里我们不能用属性 `style="width:XXX;height:XXX;"` 来定义。一个svg的js库：two.js。

## 2-2. 离线储存 applicationCache 工作原理？

- **appcache**是一种缓存机制，不是存储技术。  
在用户**没有联网**时，可以正常访问站点或应用，在用户**联网**时，更新用户机器上的缓存文件。
- **原理：**HTML5的离线存储是基于一个**新建的.appcache 文件**的缓存机制，通过这个文件上的解析离线存储资源清单，这些资源就会像 cookie 一样被存储了下来。之后当网络处于离线状态时，浏览器会通过离线存储的数据进行页面展示。
- **使用方法：**如需启用应用程序缓存，请在文档的 `<html>` 标签中包含 `<manifest>` 属性：

```
<!DOCTYPE html>
<html manifest="demo_html.appcache">
<body>
----主体内容----
</body>
</html>
```

manifest 文件的建议的文件扩展名是：".appcache"，示例如下：

```
CACHE MANIFEST
#v1.0
```

```
CACHE:  //在此标题下列出的文件将在首次下载后进行缓存
js/app.js
```

css/style.css

NETWORK: //在此标题下列出的文件需要与服务器的连接，且不会被缓存  
assets/logo.png

FALLBACK: //在此标题下列出的文件规定当页面无法访问时的回退页面（比如 404）  
/html5/ /404.html

### 2-2-1. 浏览器是怎么对 HTML5 的离线储存资源进行管理和加载的呢？

- 在线的情况下，浏览器发现 html 头部有 **manifest属性**，它会**请求 manifest 文件**
  - 如果是**第一次访问** 服务器/应用，那么浏览器就会根据 manifest 文件的内容下载相应的资源并且进行离线存储
  - 如果已经访问过服务器/应用并且资源已经离线存储了，那么浏览器就会使用**离线的资源加载页面**；然后浏览器会**对比**新的 manifest 文件与旧的 manifest 文件
    - 如果文件没有发生改变，就不做任何操作
    - 如果文件改变了，那么就会**重新下载文件中的资源**并进行离线存储
- **离线**的情况下，浏览器就直接使用离线存储的资源。在离线状态时，操作 window.applicationCache 进行需求实现。

参考链接：[HTML5 离线缓存-manifest 简介](#)

### 2-2-2. applicationCache的优点？

1. 离线浏览 - 用户可以在离线时使用应用程序
2. 快速 - 缓存的资源可以更快地加载
3. 减少服务器加载 - 浏览器只从服务器上下载已更新/已更改的资源

## 2-3. ★本地存储Cookie、LocalStorage、SessionStorage的区别？

- **携带**
  - cookie 是网站为了标示用户身份而储存在客户端上的数据（通常经过加密），会在客户端和服务器之间**来回传递**【始终在同源的 http 请求中携带（即使不需要）】，而且始终放在http 头部，较大会**影响性能**
  - SessionStorage 和 localStorage 不会自动把数据发给服务器，**仅在本地**保存。
- **存储大小**
  - cookie 数据大小不能超过 **4k**，很多浏览器都限制一个站点最多保存20个cookie。
  - sessionStorage 和 localStorage 是HTML5专门设计，存储容量比 cookie 大得多，可以达到 **5M** 或更大。
- **有效期**（生命周期）



- localStorage: **存储持久数据**，浏览器关闭后数据不丢失**除非主动删除**数据；
- sessionStorage: 数据在**当前浏览器窗口关闭**后自动删除。
- cookie: 设置的 cookie **过期时间max-age/expires**之前一直有效，即使窗口或浏览器关闭

• API易用性

- cookie需要使用document.cookie来获取和修改，而且必须使用key-value的对象格式存储。  
API简单，**需要封装**才能使用
- sessionStorage 和 localStorage 的API**简单易用**，直接使用localStorage.setItem(key,value)和localStorage.getItem(key)即可

• 共享

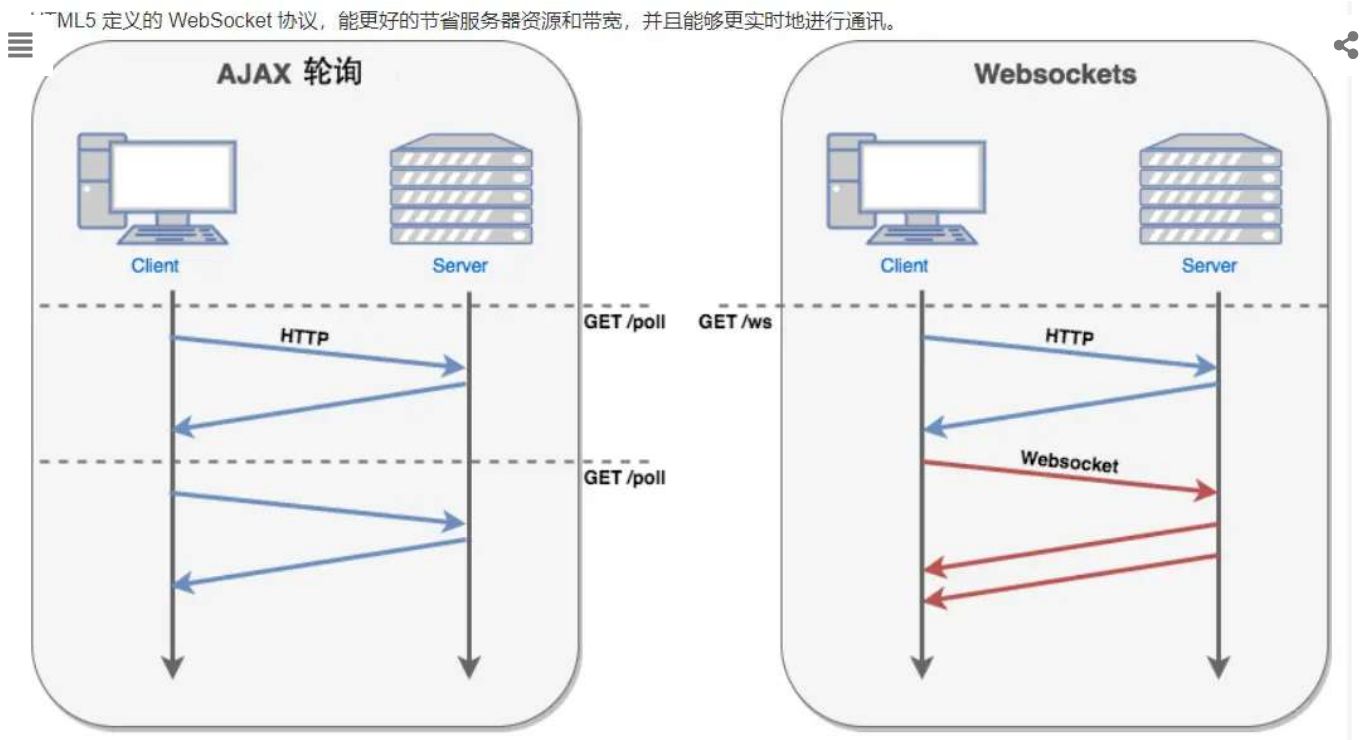
- sessionStorage**不能共享**(即使是同一个页面不同的窗口)
- localStorage在**同源文档**之间共享
- cookie在**同源且符合path**规则的文档之间共享

• 应用

- 从安全性来说，因为每次http请求都回携带cookie信息，这样浪费了带宽，所以cookie应该尽可能的少用，此外cookie还需要指定作用域，不可以跨域调用，限制很多，但是**用户识别用户**登录来说，**cookie**还是比storage好用
- 其他情况下可以用storage
  - **localStorage**可以用来在**页面内传递参数**
  - **sessionstorage**可以用来保存些**临时的数据**，防止刷新页面后丢失一些参数

API列表		举例说明
clear	清空存储中的所有本地存储数据	<code>localStorage.clear();</code>
getItem	接受一个参数key，获取对应key的本地存储	<code>localStorage.getItem('order');</code> // 对象访问方式同样有效 <code>localStorage.order = 'b110';</code> <code>localStorage.order;</code> // b110
key	接受一个整数索引，返回对应本地存储中索引的键	<code>localStorage.key(0);</code>
removeItem	接受一个参数key，删除对应本地存储的key	<code>localStorage.removeItem('order');</code>
setItem	接受两个参数，key和value，如果不存在则添加，存在则更新。	<code>localStorage.setItem('order', 'a109');</code>

2-4. WebSocket的实现和应用



- 为了建立一个 WebSocket 连接，客户端浏览器首先要向服务器发起一个 HTTP 请求，这个请求和通常的 HTTP 请求不同，包含了一些附加头信息，其中附加头信息"Upgrade: WebSocket"表明这是一个申请协议升级的 HTTP 请求。
- 服务器端解析这些附加的头信息然后产生应答信息返回给客户端，客户端和服务端端的 WebSocket 连接就建立起来了，双方就可以通过这个连接通道自由的传递信息，并且这个连接会持续存在直到客户端或者服务器端的某一方主动的关闭连接。
- 当你获取 Web Socket连接后，你可以通过`send()`方法来向服务器发送数据，并通过`onmessage`事件来接收服务器返回的数据。以下 API 用于创建 WebSocket 对象。

```
var Socket = new WebSocket(url, [protocol] );
```

#### 2-4-1. WebSocket和HTTP的区别？

- HTTP的生命周期通过Request来界定，也就是一个Request一个Response，那么在Http1.0协议中，这次Http请求就结束了。在Http1.1中进行了改进，有一个`connection: Keep-alive`，也就是说，在一个Http连接中，可以发送多个Request，接收多个Response。但是必须记住，在Http中一个Request只能对应有一个Response，而且这个Response是**被动的**，不能主动发起。
- WebSocket是基于Http协议的，或者说借用了Http协议来完成一部分握手，在握手阶段与Http是相同的。我们来看一个websocket握手协议的实现，基本是2个属性，`upgrade`，`connection`。
- WebSocket 使得客户端和服务端之间的数据交换变得更加简单，允许**服务端主动向客户端推送数据**。在 WebSocket API 中，浏览器和服务器只需要完成一次握手，两者之间就直接可以创建持久性的连接，并进行**双向数据传输**。

## 4-2. 什么是WebSocket?



- **WebSocket**是HTML5中一种在**单个 TCP 连接**上进行全双工通讯的协议，支持**持久连接**；http协议不支持持久性连接。HTTP1.0和HTTP1.1都不支持持久性的连接，HTTP1.1中的keep-alive，**将多个http请求在1个TCP连接上实现**。
- 应用：**实时通信应用**[微信网页版]

### 2-4-3. WebSocket具体有什么优点?

- WebSocket 使得客户端和服务端之间的数据交换变得更加简单，允许**服务端主动向客户端推送数据**。在 WebSocket API 中，浏览器和服务器只需要完成一次握手，两者之间就直接可以创建持久性的连接，并进行**双向数据传输**。

基本请求如下：

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket //表示客户端希望升级到 websocket 协议
Connection: Upgrade //表示客户端希望连接升级
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

- 多了下面2个属性：
  1. Upgrade:websocket
  2. Connection:Upgrade
- 告诉服务器发送的是 **websocket**
  - Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
  - Sec-WebSocket-Protocol: chat, superchat
  - Sec-WebSocket-Version:13

### 2-4-4. WebSocket如何兼容低版本浏览器?

- 使用**flash**或其他方法实现一个websocket客户端（Adobe Flash Socket）；
- 基于**长轮询/长连接**的 XHR；
- ActiveX HTMLFile (IE)；
- 基于 multipart 编码发送 XHR；
  - **轮询**：客户端定时向服务器发送Ajax请求，服务器接到请求后马上返回响应信息并关闭连接。





- 优点：后端程序编写比较容易。
- 缺点：请求中有大半是无用，浪费带宽和服务器资源。
- 实例：适于小型应用。
- **长轮询**：客户端向服务器发送Ajax请求，**服务器接到请求后hold住连接，直到有新消息才返回响应信息并关闭连接**，客户端处理完响应信息后再向服务器发送新的请求。
  - 优点：在无消息的情况下不会频繁的请求，耗费资源小。
  - 缺点：服务器hold连接会消耗资源，返回数据**顺序无保证**，难于管理维护。
  - 实例：WebQQ、Hi网页版、Facebook IM。
- **长连接**：在页面里嵌入一个隐藏iframe，将这个隐藏iframe的src属性设为对一个长连接的请求或是采用xhr请求，服务器端就能源源不断地往客户端输入数据。
  - 优点：消息即时到达，不发无用请求；管理起来也相对方便。
  - 缺点：服务器维护一个长连接会增加开销。
  - 实例：**Gmail聊天**
- **Flash Socket**：在页面中内嵌入一个使用了Socket类的 Flash 程序，JavaScript通过调用此Flash程序提供的Socket接口与服务器端的Socket接口进行通信，JavaScript在收到服务器端传送的信息后控制页面的显示。
  - 优点：实现真正的即时通信，而不是伪即时。
  - 缺点：**客户端必须安装Flash插件**；非HTTP协议，无法自动穿越防火墙。
  - 实例：网络互动游戏。

## 2-5. WebWorker的实现和应用

### 2-5-1. WebWorker是什么，有什么优缺点？

**web worker**是运行在**后台**的 **JavaScript**，不会影响页面的性能。

- 当在 HTML 页面中执行普通脚本时，页面的状态是不可响应的，直到脚本已完成。
- web worker 是运行在**后台**的 JavaScript，独立于其他脚本，不会影响页面的性能。
- Web Worker 的作用，就是为 JavaScript 创造**多线程环境**，允许主线程创建 Worker 线程，将一些任务分配给后者运行。在主线程运行的同时，Worker 线程在后台运行，两者**互不干扰**。等到 Worker 线程完成计算任务，再把结果返回给主线程。
- 优点：
  - 一些计算**密集型或高延迟**的任务，被 Worker 线程负担了，**主线程**（通常负责 UI 交互）就会很流畅，**不会被阻塞**或拖慢。
  - **Worker 线程**一旦新建成功，就会始终运行，**不会被主线程**上的活动（比如用户点击按钮、提交表单）**打断**。这样**有利于随时响应主线程的通信**。
- 缺点：

- 但是，这也造成了 Worker 比较**耗费资源**，不应该过度使用，而且一旦使用完毕，就应该关闭。

## 2-5-2. WebWorker的创建方法?

主要是**postMessage**和**onmessage**

- **检测浏览器的支持性** `typeof Worker !== "undefined"`
- **创建**demo\_worker.js文件，传送消息 `postMessage('消息内容');`
- **创建实例化对象** `w = new Worker("demo_worker.js")`
- **接收消息** `w.onmessage = function(event){//TODO}`
- **终止Webworker** `w.terminate()`

## 2-5-3. WebWorker使用注意事项?

1. **同源限制** 分配给 Worker 线程运行的脚本文件，必须与**主线程的脚本文件同源**。
2. **DOM 限制** Worker 线程所在的全局对象，与主线程不一样，无法读取主线程所在网页的 DOM 对象，也**无法使用** `document` 、 `window` 、 `parent` **这些对象**。但是，Worker 线程可以 `navigator` 对象和 `location` 对象。
3. **通信联系** Worker 线程和主线程不在同一个上下文环境，它们不能直接通信，必须**通过消息完成**。
4. **脚本限制** Worker 线程不能执行 `alert()` 方法和 `confirm()` 方法，但可以使用 XMLHttpRequest 对象发出 **AJAX** 请求。
5. **文件限制** Worker 线程无法读取本地文件，即不能打开本机的文件系统 ( `file://` ) ，它所加载的脚本，**必须来自网络**。

## 2-6. SharedWorker是什么?

**共享worker(SharedWorker)**，这种是可以多个标签页、iframe共同使用的，可以实现标签页之间的通信。SharedWorker可以被多个window共同使用，但必须保证这些标签页都是同源的(相同的协议，主机和端口号)

**举例1**：但是不支持本地运行，需放到服务器。

### 2-6-1. SharedWorker的创建方法?

- **检测浏览器的支持性** `typeof Worker !== "undefined"`
- 创建worker.js文件，监听消息事件 `onmessage`
  - 消息里有 `get` 就传送消息 `postMessage('消息内容')` 到客户端
  - 否则就把客户端传递过来的数据存储在data变量
- **创建对象** `w = new SharedWorker("demo_worker.js")`
- **请求**使用postMessage来传递发送数据的请求data以及获取数据的请求ge



- ☰ 页面A发送数据给worker，然后打开页面B
  - 调用 `window.worker.port.postMessage('get')`，即可收到页面A发送给worker的数据
  - 调用 `window.worker.port.postMessage('发送内容')`，即可传送数据到sharedWorker

## 2-7. drag API包含哪些？

1. **dragstart**: 事件主体是被拖放元素，在**开始拖放**被拖放元素时触发，。
2. **drag**: 事件主体是被拖放元素，在**正在拖放**被拖放元素时触发。
3. **dragenter**: 事件主体是目标元素，在被拖放元素**进入某元素**时触发。
4. **dragover**: 事件主体是目标元素，在被拖放**在某元素内移动**时触发。
5. **dragleave**: 事件主体是目标元素，在被拖放元素**移出目标元素**是触发。
6. **drop**: 事件主体是目标元素，在**目标元素完全接受**被拖放元素时触发(完全进入)。
7. **dragend**: 事件主体是被拖放元素，在整个**拖放操作结束**时触发。

- 具体使用:

```
cat.ondragstart = function(e){
    console.log('cat开始移动');
    offsetX = e.offsetX;
    offsetY = e.offsetY;
};
```

## 2-8. Geolocation地图定位

```
navigator.geolocation.getCurrentPosition(showPosition,showError)
```

- 实例流程
  - 检测是否支持地理定位
  - 如果支持，则运行 **getCurrentPosition()方法**。如果不支持，则向用户显示一段消息。
  - 如果getCurrentPosition()运行成功，则向参数showPosition中规定的函数返回一个coordinates对象
  - **showPosition() 函数**获得并显示经度和纬度

## 2-9. 页面可见性（Page Visibility API） 可以有哪些用途？

- 通过 `visibilityState` 的值检测页面当前是否可见，以及打开网页的时间等；
- 在页面被切换到其他后台进程的时候，自动暂停音乐或视频的播放；
- 参考

## 3. 应用？



## 1. 如何实现浏览器内多个标签页之间的通信?

- **WebSocket、SharedWorker**; 【postMessage、onmessage】
- 也可以调用 **localStorage、cookies** 等本地存储方式; **localStorage** 另一个浏览上下文里被添加、修改或删除时, 它都会触发一个 **storage** 事件, 我们通过监听事件, 控制它的值来进行页面信息通信; **“当同源页面的某个页面修改了localStorage, 其余的同源页面只要注册了storage事件, 就会触发”** 所以, **localStorage** 的例子运行需要如下条件:
  - 同一浏览器打开了两个同源页面
  - 其中一个网页修改了 **localStorage**
  - 另一网页注册了 **storage** 事件

### 1. 在同源的两个页面中, 可以**监听 storage 事件**

```
window.addEventListener("storage", function (e) {  
    alert(e.newValue);  
});
```

### 2. 在同一个页面中, 对 **localStorage** 的 **setItem** 方法进行重写

```
var originalSetItem = localStorage.setItem;  
localStorage.setItem = function(key,newValue){  
    var setItemEvent = new Event("setItemEvent");  
    setItemEvent.newValue = newValue;  
    window.dispatchEvent(setItemEvent);  
    originalSetItem.apply(this,arguments);  
}  
window.addEventListener("setItemEvent", function (e) {  
    alert(e.newValue);  
});  
localStorage.setItem("name", "wang");
```

**注意** quirks: Safari 在无痕模式下设置 **localstorge** 值时会抛出 **QuotaExceededError** 的异常。

## 3-2. 如何实现实时通信?

- 现在, 很多网站为了实现推送技术, 所用的技术都是 **Ajax 轮询**。轮询是在特定的时间间隔 (如每1秒), 由浏览器对服务器发出**HTTP请求**, 然后由服务器返回最新的数据给客户端的浏览器。这种**传统的模式**带来很明显的缺点, 即**浏览器需要不断的向服务器发出请求**, 然而**HTTP请求**可能包含较长的头部, 其中真正有效的数据可能只是很小的一部分, 显然这样会**浪费很多的带宽等资源**。

☰ HTML5 定义的 WebSocket 协议支持服务端push，能更好的节省服务器资源和带宽，并且能够实时地进行通讯。

- 因此我们可以通过WebSocket实例化对象，上面挂载的onopen方法建立连接时触发，**send**方法发送数据时触发，**onmessage**方法在接收数据时触发，onerror/onclose...

上一篇：[一、HTML&CSS](#)

下一篇：[2.浏览器](#)