HTMl5的存储方式sessionStorage和localStorage详解

html5中的Web Storage包括了两种存储方式：sessionStorage和localStorage。sessionStorage用于本地存储一个会话（session）中的数据，这些数据只有在同一个会话中的页面才能访问并且当会话结束后数据也随之销毁

因此sessionStorage不是一种持久化的本地存储，仅仅是会话级别的存储。而localStorage用于持久化的本地存储，除非主动删除数据，否则数据是永远不会过期的。  
一、web storage和cookie的区别  
Web Storage的概念和cookie相似，区别是它是为了更大容量存储设计的。Cookie的大小是受限的，并且每次你请求一个新的页面的时候Cookie都会被发送过去，这样无形中浪费了带宽，另外cookie还需要指定作用域，不可以跨域调用。  
除此之外，Web Storage拥有setItem,getItem,removeItem,clear等方法，不像cookie需要前端开发者自己封装setCookie，getCookie。  
但是Cookie也是不可以或缺的：Cookie的作用是与服务器进行交互，作为HTTP规范的一部分而存在 ，而Web Storage仅仅是为了在本地“存储”数据而生

二、html5 web storage的浏览器支持情况  
浏览器的支持除了IE７及以下不支持外，其他标准浏览器都完全支持(ie及FF需在web服务器里运行)，值得一提的是IE总是办好事，例如IE7、IE6中的UserData其实就是javascript本地存储的解决方案。通过简单的代码封装可以统一到所有的浏览器都支持web storage。  
要判断浏览器是否支持localStorage可以使用下面的代码：

**if**(**window**.**localStorage**){  
 alert(**"浏览支持localStorage"**)  
}**else** {  
 alert(**"浏览暂不支持localStorage"**)  
}  
*//或者***if**(typeof **window**.**localStorage** == **'undefined'**){  
 alert(**"浏览暂不支持localStorage"**)  
}

三、localStorage和sessionStorage操作  
localStorage和sessionStorage都具有相同的操作方法，例如setItem、getItem和removeItem等  
localStorage和sessionStorage的方法：  
setItem存储value  
用途：将value存储到key字段  
用法：.setItem( key, value)  
代码示例：

**sessionStorage**.setItem(**"key"**, **"value"**);  
**localStorage**.setItem(**"site"**, **"js8.in"**);

getItem获取value  
用途：获取指定key本地存储的值  
用法：.getItem(key)  
代码示例：

**var *value*** = **sessionStorage**.getItem(**"key"**);  
 **var *site*** = **localStorage**.getItem(**"site"**);  
</**script**>

removeItem删除key  
用途：删除指定key本地存储的值  
用法：.removeItem(key)  
代码示例：

**sessionStorage**.removeItem(**"key"**);  
**localStorage**.removeItem(**"site"**);

clear清除所有的key/value  
用途：清除所有的key/value  
用法：.clear()  
代码示例：

**sessionStorage**.clear();  
**localStorage**.clear();

四、其他操作方法：点操作和[]  
web Storage不但可以用自身的setItem,getItem等方便存取，也可以像普通对象一样用点(.)操作符，及[]的方式进行数据存储，像如下的代码：

**var *storage*** = **window**.**localStorage**; ***storage***.**key1** = **"hello"**;  
***storage***[**"key2"**] = **"world"**;  
**console**.log(***storage***.**key1**);  
**console**.log(***storage***[**"key2"**]);

五、localStorage和sessionStorage的key和length属性实现遍历  
sessionStorage和localStorage提供的key()和length可以方便的实现存储的数据遍历，例如下面的代码：

**var *storage*** = **window**.**localStorage**;  
**for** (**var *i***=0, ***len*** = ***storage***.**length**; ***i*** < ***len***; ***i***++) {  
 **var *key*** = ***storage***.key(***i***);  
 **var *value*** = ***storage***.getItem(***key***);  
 **console**.log(***key*** + **"="** + ***value***);  
}

六、storage事件  
storage还提供了storage事件，当键值改变或者clear的时候，就可以触发storage事件，如下面的代码就添加了一个storage事件改变的监听：

**if**(**window**.addEventListener){  
 **window**.addEventListener(**"storage"**,*handle\_storage*,**false**);  
} **else if**(**window**.attachEvent) {  
 **window**.attachEvent(**"onstorage"**,*handle\_storage*);  
}  
**function** *handle\_storage*(e){  
 **if**(!e){  
 e=**window**.**event**;  
 }  
}

HTML5之Web Worker

Web Worker是什么：

web worker是[HTML5](http://lib.csdn.net/base/html5)提供的一个多线程解决方案，（以下我们简称wWorker），大家都知道js是单线程执行的，所谓多线程解决方案也就是不需要js在执行的时候摁着一个线程可劲使用，比如有些时候我们不小心会弄出死循环 或者大量计算时候，浏览器就会在它执行的线程中去执行我们的计算，从而导致ui渲染线程挂起浏览器就over了，这就是单线程处理的结果，多线程就是解决了这么一个问题。在之前我们利用js来模拟多线程一般都是用setTimeOut延迟处理时间来达到”多线程“处理效果。但是这种处理都是手动规定其闲余时间去处理内部的函数，我们并不知道到底处理了多久 或者应该处理多久，只能去猜测。假如服务端每隔两秒向里面更新一下数据，而setTimeOut执行事件为3秒，这样就会有数据处理丢失问题了。并且用setTimeout模拟起来计算时间模拟比较复杂wWorker在浏览器中开辟一个新线程，将需要大量计算的或者容易导致阻塞的程序放置于另一个线程中，而其他执行的东东还是依然在本线程中工作。这样一来大大提高了浏览器运行效率。[Html5](http://lib.csdn.net/base/html5)规范了wWorker三大特性：

1.能够长时间运行（响应）；

2.理想的启动性能以及理想的内存消耗；

3.wWorkers 允许开发人员编写能够长时间运行而不被用户所中断的后台程序，去执行事务或者逻辑，并同时保证页面对用户的及时响应。

wWorker虽然不能直接处理dom元素（毕竟不是实际js脚本）但是可以创建数据消息来通过主线程的回调函数去执行dom。

Web Worker如何使用

举个栗子：

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Title</**title**>  
 <**script**>  
 *//WEB页主线程* **var *worker*** =**new** Worker(**"worker.js"**); *//创建一个线程对象并向它传递将在新线程中执行的脚本的URL* ***worker***.postMessage(**"hello wWorker!"**); *//向新线程发送数据* ***worker***.onmessage =**function**(evt){ *//接收worker传过来的数据函数* **console**.log(evt.data); *//执行worker发送来的数据* }  
 </**script**>  
</**head**>  
<**body**>  
   
</**body**>  
</**html**>

worker.js(新开辟线程)

onmessage=**function**(event){  
 postMessage(event.data);*//发送给主线程接受到数据（‘hello wWorker!’）*}

在web环境下执行html代码后会在控制台输出hello wWorker!

分析一下整个应用执行过程：

1.在主线程实例化一个worker并且加载一个js的url地址作为参数

2.通过worker.postMessage向新线程发送一个参数或者数据

3.新线程绑定onmessage处理主线程发送过来的数据

4.新线程处理完主线程发来数据后利用postMessage方法向主线程发送数据

5.主线程绑定onmessage处理新线程返回的数据

Web Worker能做什么

我们知道应用该方法可以通过开辟一个新线程来协助处理一个主线程的复杂工作。

举个例子（来自互联网）：

fibonacci数列在数学上以递归方法定义为：F0=0，F1=1，Fn=F(n-1)+F(n-2)（n>=2，n∈N\*）

而javascript的常用实现为：

<**script**>  
 **var** *fibonacci* =**function**(n) {  
 **return** n <2? n : arguments.**callee**(n -1) + arguments.**callee**(n -2);  
 };  
</**script**>

因为js执行计算是单线程的，在计算如此[大数据](http://lib.csdn.net/base/hadoop)量时候 浏览器往往就会被挂起，从而导致页面假死状态。实则是浏览器线程在后台进行计算，在这个计算过程中页面是不会继续向下进行渲染等操作的，所以我们可以通过worker将计算放入一个新的线程中去，从而避免单线程工作阻塞浏览器继续执行。

举个栗子：

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Title</**title**>  
 <**script**>  
 **window**.onload =**function**(){  
 **var** timer = (**new** Date()).valueOf();  
 **var** worker =**new** Worker(**'worker.js'**);  
 worker.postMessage(40);  
 worker.addEventListener(**'message'**, **function**(event) {  
 **var** timer2 = (**new** Date()).valueOf();  
 **console**.log( **'结果：'**+event.data+**'用时：'**+ ( timer2-timer));  
 }, **false**);  
 }  
 </**script**>  
</**head**>  
<**body**>  
  
</**body**>  
</**html**>

新线程worker.js

<**script**>  
 **var** *fibonacci* =**function**(n) {  
 **return** n <2? n : arguments.**callee**(n -1) + arguments.**callee**(n -2);  
 };  
 onmessage =**function**(event) {  
 **var** n = parseInt(event.data, 10);  
 postMessage(*fibonacci*(n));  
 };  
</**script**>

在控制台我们可以看到

结果：102334155用时：9452 ；

web worker总结：  
优点：

可以加载一个JS进行大量的复杂计算而不挂起主进程，并通过postMessage，onmessage进行通信

局限性：

1.不能跨域加载JS

2.worker内代码不能访问DOM

3.各个浏览器对Worker的实现不大一致，例如FF里允许worker中创建新的worker,而Chrome中就不行

综上所述：web worker目前所能做的(应该说是最适合的)也就是新开辟一个线程去计算大数据或者处理并发事件，而不影响主线程进程。