# HTML5 第六天笔记

## Web Workers

### 1. 基础内容

#### 1）什么是 Web Workers

能够把 JavaScript 计算委托给后台线程，通过允许这些活动以防止使交互型事件变得缓慢。

上句话是 MDN 对象 Web Workers 的简单描述。

Web Workers 是可以在后台（页面端）运行的任务，它能够被轻松的创建，还能向它的创建者发送消息。

Web Workers 的三大主要特征：能够长时间运行（响应），理想的启动性能以及理想的内存消耗。

#### 2）两种 Web Workers

Web Workers 可以分为两种类型：Dedicated Web Worker（专用线程）和 Shared Web Worker（共享线程）。

##### a. Dedicated Web Worker

目前绝大多数的应用场景所使用的是 Dedicated Web Worker（专用线程）。Dedicated Web Worker随着 HTML 页面关闭而被终止，这就意味着 Dedicated Web Worker 只能被创建它的 HTML 页面所访问。

在 JavaScript 代码中，Work 类型代表 Dedicated Web Worker。

##### b. Shared Web Worker

与 Dedicated Web Worker（专用线程）相比，Shared Web Worker 一般用于一些特定情况，可以为多个 HTML 页面服务，即可以被多个 HTML 页面所访问。与之对应的，只有将与之相关的所有 HTML 页面都关闭才能被终止。

在 JavaScript 代码中，SharedWorker 类型代表 Shared Web Worker。

#### 3）Web Workers 的限制

Web Workers 在具体使用中，具有以下几种限制：

* Web Workers 无法访问 DOM 节点；
* Web Workers 无法访问全局变量或是全局函数；
* Web Workers 无法调用 alert() 或者 confirm() 之类的函数；
* Web Workers 无法访问 window、document 之类的浏览器全局变量；

Web Workers 可以在它的作用域内访问 navigator 对象。它含有如下能够识别浏览器的字符串，就像在普通脚本中做的那样：

* appName
* appVersion
* platform
* userAgent

#### 4）Web Workers 的使用场景

* Web Workers 线程能够在不干扰 UI 的情况下执行一些计算的任务。
* Web Workers 线程能够使用 XMLHttpRequest 对象与服务器端进行异步交互。

#### 5）参考资料

* [Web Workers - W3C](http://www.w3.org/TR/workers/)
* [使用 Web Workers](https://developer.mozilla.org/zh-CN/docs/Web/API/Web_Workers_API/Using_web_workers)
* [Microsoft Web Workers](https://msdn.microsoft.com/library/hh673568(v=vs.85).aspx)

### 2. Worker API

#### 1）Worker 构造函数

Worker(DOMString scriptURL)

通过该构造函数可以直接创建 Worker 对象，并传递 Worker 文件的路径（scriptURL）。具体用法如下：

var worker = new Worker(my\_task.js);

通过 Worker 的构造函数得到 Worker 对象，在 HTML 页面与 Worker 文件之间创建通信。

**值得注意的是：**指定的 Worker 文件路径必须遵循同源策略。

#### 2）监听通信消息机制

worker.onmessage = function(event){

逻辑编写在这里...

}

当 Worker 传递消息给 HTML 页面（或 HTML 页面传递消息给 Worker）时，可以通过 onmessage 事件进行监听并且捕获。

传递的消息存储在 onmessage 事件处理函数的参数 data 中。

worker.onmessage = function(event) {

console.log("Worker's message: " + event.data);

};

#### 3）发送通信消息方法

postMessage(JSObject message)

HTML 页面通过该方法向 Worker 的内部作用域内传递消息。该方法接收一个单独的参数，即要传递给 Worker 的数据。

message 参数用于传递给 Worker 的数据，该数据将包含于传递给 onmessage 处理函数的事件对象中的 data 字段内。

// 普通数据

worker.postMessage("ali");

// JSON格式的数据

worker.postMessage({"cmd": "init", "timestamp": Date.now()});

#### 4）终止 Worker 通信

terminate()

该方法用于立即终止 Worker。该方法不会给 Worker 留下任何完成操作的机会；就是简单的立即停止。

#### 5）监听通信错误机制

worker.onerror = function(event){

错误处理编写在这里...

}

错误信息对象包含三个属性：

* message：一个可读性良好的错误信息。
* filename：产生错误的脚本文件名。
* lineno：发生错误时所在的脚本文件行号。

### 3. 双向通信

所谓“单向通信”就是指 HTML 页面与 Worker 建立通信后，只是 Worker 向 HTML 页面传递消息。

所谓“双向通信”就是指 HTML 页面与 Worker 之间可以相互传递消息的一种方式。

要建立双向通信，HTML 页面和 Worker 线程都要侦听 onmessage 事件。

* 首先，该脚本创建 worker 线程。

var echo = new Worker('echo.js');

echo.onmessage = function(e) {

alert(e.data);

}

* 当用户单击提交按钮时，脚本会将两条信息以 JavaScript 对象文本的形式传递给 Worker。

<script>

window.onload = function() {

var echoForm = document.getElementById('echoForm');

echoForm.addEventListener('submit', function(e) {

echo.postMessage({

message : e.target.message.value,

timeout : e.target.timeout.value

});

e.preventDefault();

}, false);

}

</script>

<form id="echoForm">

<p>Echo the following message after a delay.</p>

<input type="text" name="message" value="Input message here."/><br/>

<input type="number" name="timeout" max="10" value="2"/> seconds.<br/>

<button type="submit">Send Message</button>

</form>

* 最后，Worker 侦听消息，并在指定的超时间隔之后将其返回。

onmessage = function(e){

setTimeout(function(){

postMessage(e.data.message);

},

e.data.timeout \* 1000);

}

## Web 存储

在开发 Web 应用时，开发人员有时需要在本地存储数据。当前浏览器支持 cookie 存储，但其大小有 4KB 的限制。

HTML5 中新引入了 Web Storage 机制，通过使用键值对在客户端保存数据，并且提供了更大容量的存储空间。

HTML5 中的 Web 存储对象有两种类型：

* sessionStorage 对象负责存储一个会话的数据。如果用户关闭了页面或浏览器，则会销毁数据。
* localStorage 对象负责存储没有到期的数据。当 Web 页面或浏览器关闭时，仍会保持数据的存储，当然这还取决于为此用户的浏览器设置的存储量。

这两种存储对象具有相同的方法和属性。

**值得注意的是：**Web 存储并不比 cookies 安全。所以不要在客户端存储敏感信息，比如密码或信用卡信息。

官方规范资料地址：<https://html.spec.whatwg.org/multipage/webstorage.html>

### 1. sessionStorge

#### 1）浏览器支持性检查

//sessionStorage

if(window.sessionStorage){

alert(“support sessionStorage”);

}else{

alert(“not support sessionStorage”);

// 不支持 sessionStorage

}

#### 2）sessionStorage的方法

| **方法名称** | **描述** |
| --- | --- |
| setItem(key, value) | 为 Web 存储对象添加一个键/值对，供以后使用。该值可以是任何的数据类型：字符串、数值、数组等。 |
| getItem(key) | 基于起初用来存储它的这个键检索值。 |
| clear() | 从此 Web 存储对象清除所有的键/值对。 |
| removeItem(key) | 基于某个键从此 Web 存储对象清除特定的键/值对。 |
| key(n) | 检索 key[n] 的值。 |

##### a. 向 Web 存储对象添加键/值对

* 使用 setItem() 方法

sessionStorage.setItem('myKey', 'myValue');

* 使用 Web 存储对象来直接设置此键的值

sessionStorage.myKey = 'myValue';

##### b. 从 Web 存储对象中检索键/值对

* 使用 getItem() 方法

sessionStorage.getItem('myKey');

* 使用 Web 存储对象来直接获取此键的值

sessionStorage.myKey;

##### c. 删除所有键/值对

sessionStorage.clear();

##### d. 删除单个键/值对

sessionStorage.removeItem('myKey');

#### 示例代码

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>sessionStorage</title>

</head>

<body>

<input type="text" id="data">

<input type="button" id="save" value="保存">

<input type="button" id="read" value="读取">

<input type="button" id="dele" value="删除">

<script>

var save = document.getElementById("save");

save.onclick = function(){

// 将用户输入的数据,保存到sessionStorage中

// 1. 获取用户输入的数据

var value = document.getElementById("data").value;

// 2. 如何生成存储数据所使用的KEY

var key = new Date().getTime();// 时间戳

// 3. 存储数据

window.sessionStorage.setItem(key,value);

}

var read = document.getElementById("read");

read.onclick = function(){

// 读取sessionSotorage中所有的数据

// 1. 获取当前sessionStorage中所有数据的个数

var count = window.sessionStorage.length;

// 2. 进行遍历

for(var i=0;i<count;i++){

// 3. 根据索引值得到key

var key = window.sessionStorage.key(i);

// 4. 根据key值得到value

var value = window.sessionStorage.getItem(key);

// 5. 进行测试

console.log(key + " : " + value);

}

}

var dele = document.getElementById("dele");

dele.onclick = function(){

// removeItem()

//window.sessionStorage.removeItem(1452666425707);

window.sessionStorage.clear();

}

</script>

</body>

</html>

### 2. localStorage

#### 1）浏览器支持性检查

//localStorage

if(window.localStorage){

alert(“support localStorage”);

}else{

alert(“not support localStorage”);

// 不支持 localStorage

}

#### 2）localStorage的方法

| **方法名称** | **描述** |
| --- | --- |
| setItem(key, value) | 为 Web 存储对象添加一个键/值对，供以后使用。该值可以是任何的数据类型：字符串、数值、数组等。 |
| getItem(key) | 基于起初用来存储它的这个键检索值。 |
| clear() | 从此 Web 存储对象清除所有的键/值对。 |
| removeItem(key) | 基于某个键从此 Web 存储对象清除特定的键/值对。 |
| key(n) | 检索 key[n] 的值。 |

##### a. 向 Web 存储对象添加键/值对

* 使用 setItem() 方法

localStorage.setItem('myKey', 'myValue');

* 使用 Web 存储对象来直接设置此键的值

localStorage.myKey = 'myValue';

##### b. 从 Web 存储对象中检索键/值对

* 使用 getItem() 方法

localStorage.getItem('myKey');

* 使用 Web 存储对象来直接获取此键的值

localStorage.myKey;

##### c. 删除所有键/值对

localStorage.clear();

##### d. 删除单个键/值对

localStorage.removeItem('myKey');

#### 3）localStorage的事件

如果想在存储成功或修改存储的值时执行一些操作，可以用 Web Storage 接口提供的事件。

// 显示存储事件的相关内容

function handleStorageEvent(e) {

document.write(“key” + e.key + “oldValue” + e.oldValue + “newValue” + e.newValue);

}

// 添加存储事件监听

window.addEventListener(“storage”, handleStorageEvent, false);

| **事件参数** | **描述** |
| --- | --- |
| key | 发生改变的键 |
| oldValue | 键改变之前的值 |
| newValue | 键改变之后的值 |
| url | 触发存储事件的页面 url |

## Web Sockets

### 1. 什么是 Web Sockets

#### 1）什么是 Web Sockets

Web Sockets 技术使得浏览器直接与服务器端的程序通过 socket 可以实时的推送或者获取信息的通讯方式成为可能。

在 HTML5 之前实现浏览器与服务器端实时通信的技术如下：

* 轮询：原理简单易懂，就是客户端通过一定的时间间隔以频繁请求的方式向服务器发送请求，来保持客户端和服务器端的数据同步。
* Flash：Flash 通过自己的 Socket 实现完成数据交换，再利用 Flash 暴露出相应的接口为 JavaScript 调用，从而达到实时传输目的。

#### 2）Web Sockets 的优势与劣势

##### a. Web Sockets 的优势

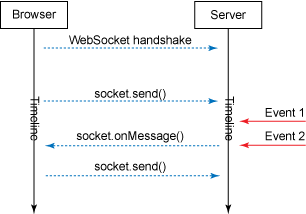
* Web Sockets 提供强大的、双向、低延迟和易于处理的错误。
* 它没有很多连接，比如：Comet 长轮询。

##### b. Web Sockets 的劣势

* Web Sockets 是 HTML5 的新规范，并不是所有浏览器都支持。
* 无请求作用域。由于 Web Sockets 是一个 TCP 套接字，而不是一个 HTTP 请求，因此无法轻松使用请求作用域服务。

#### 3）Web Sockets 的实现原理

下图展示了如何使用 WebSockets 进行通信：



1. HTTP 握手被发送到带有特定标头的服务器。
2. 在 JavaScript 的服务器或客户端上提供某种类型的套接字。
3. 使用该套接字来通过事件处理器异步接收数据。

### 2. Web Sockets API

#### 1）Web Sockets 的构造函数

var websocket = new WebSocket(url, protocols);

| **参数名称** | **描述** |
| --- | --- |
| url | 表示要连接的URL,这个URL应该是响应WebSocket的地址。 |
| protocols | 单个协议名称或字符串数组,默认为空字符串。 |

**值得注意的是：**Web Sockets 的请求地址（url）必须是 ws:// 或 wss://。

var exampleSocket = new WebSocket("ws://www.example.com/socketserver");

#### 2）Web Sockets 的方法

| **方法名称** | **参数** | **描述** |
| --- | --- | --- |
| send(data) | data表示发送的请求数据。 | 通过 WebSocket 连接向服务器发送数据。 |
| close() |  | 关闭WebSocket连接或停止正在进行的连接请求。 |

exampleSocket.send("Here's some text that the server is urgently awaiting!");

#### 3）Web Sockets 的属性

Web Sockets 的属性主要使用的就是 readyState，该属性表示连接的当前状态。

| **常量名** | **值** | **描述** |
| --- | --- | --- |
| CONNECTING | 0 | 连接还没开启。 |
| OPEN | 1 | 连接已开启并准备好进行通信。 |
| CLOSING | 2 | 连接正在关闭的过程中。 |
| CLOSED | 3 | 连接已经关闭，或者连接无法建立。 |

#### 4）Web Sockets 的事件

| **事件名称** | **描述** |
| --- | --- |
| onopen | 用于监听 Web Sockets 打开的事件。 |
| onmessage | 用于监听 Web Sockets 服务器端传递消息的事件。 |
| onerror | 用于监听 Web Sockets 发生错误的事件。 |
| onclose | 用于监听 Web Sockets 通信关闭的事件。 |

#### 5）Web Sockets 使用步骤

var ws = new WebSocket('ws://127.0.0.1:8080/async');

ws.onopen = function() {

// called when connection is opened

};

ws.onerror = function(e) {

// called in case of error, when connection is broken in example

};

ws.onclose = function() {

// called when connexion is closed

};

ws.onmessage = function(msg) {

// called when the server sends a message to the client.

// msg.data contains the message.

};

// Here is how to send some data to the server

ws.send('some data');

// To close the socket:

ws.close();

### 3. Web Sockets 案例

* 客户端代码示例：

// 假设服务端ip为127.0.0.1

ws = new WebSocket("ws://127.0.0.1:2346");

ws.onopen = function() {

alert("连接成功");

ws.send('tom');

alert("给服务端发送一个字符串：tom");

};

ws.onmessage = function(e) {

alert("收到服务端的消息：" + e.data);

};

* 服务器端代码示例：

这里利用了 PHP 的第三方框架 Workerman 实现 Web Sockets 的服务器端逻辑。

<?php

use Workerman\Worker;

require\_once '/Workerman/Autoloader.php';

// 创建一个Worker监听2346端口，使用websocket协议通讯

$ws\_worker = new Worker("websocket://0.0.0.0:2346");

// 启动4个进程对外提供服务

$ws\_worker->count = 4;

// 当收到客户端发来的数据后返回hello $data给客户端

$ws\_worker->onMessage = function($connection, $data)

{

// 向客户端发送hello $data

$connection->send('hello ' . $data);

};

// 运行worker

Worker::runAll();

?>

**原创声明：**如发布或转载请注明以下作者信息，谢谢！

本文作者 - 金云龙

联系方式 - QQ：2080954535；微信：imlonge；微信公众号：longestory

个人网站：<http://www.longestory.com>