HTML5之多线程(Web Worker)

作者：有故事的人 来源：范文网 www.wtabcd.cn/fanwen/

本文原地址：https://www.wtabcd.cn/fanwen/zuowen/af7ae57f183ae4ec4546469f25558b48.html

提到 html5 总是让人津津乐道，太多的特性和有趣的 api
让人耳目一新。但是很多童鞋还停留在语义化的阶段，忽视了 html5 的强劲之处。

这节我们来探讨一下多线程 web-worker。

一、明确 javascript 是单线程

javascript 语言的一大特点就是单线程，也就是说，同一个时间只能做一件事。

听起来有些匪夷所思，为什么不设计成多线程提高效率呢？我们可以假设一种场景：

假定 javascript 同时有两个线程，一个线程在某个 dom
节点上添加内容，另一个线程删除了这个节点，这时浏览器应该以哪个线程为准？

作为浏览器脚本语言， javascript 的主要用途是与用户互动，以及操作 dom 。

这决定了它只能是单线程，否则会带来很复杂的同步问题。为了避免复杂性，从一诞生，
javascript 就是单线程，这已经成了这门语言的核心特征，估计短期内很难改变。

二、新曙光：web worker

单线程始终是一个痛点，为了利用多核 cpu 的计算能力， html5 提出 web worker 标准，允许
javascript 脚本创建多个线程。但是子线程完全受主线程控制，且不得操作 dom 。

所以，这个新标准并没有改变 javascript 单线程的本质。

web workers 是现代浏览器提供的一个 javascript 多线程解决方案，我们可以找到很多使用场景：

1.我们可以用 web worker 做一些大计算量的操作；

2.可以实现轮询，改变某些状态；

3.页头消息状态更新，比如页头的消息个数通知；

4.高频用户交互，拼写检查，譬如：根据用户的输入习惯、历史记录以及缓存等信息来协助用户完成输入的纠错、校正功能等

5.加密：加密有时候会非常地耗时，特别是如果当你需要经常加密很多数据的时候（比如，发往服务器前加密数据）。

6.预取数据：为了优化网站或者网络应用及提升数据加载时间，你可以使用 workers

来提前加载部分数据以备不时之需。

加密是一个使用 web worker 的绝佳场景，因为它并不需要访问 dom 或者利用其它魔法，它只是纯粹使用算法进行计算而已。随着大众对个人敏感数据的日益重视，信息安全和加密也成为重中之重。这可以从近期的 12306 用户数据泄露西安工大事件中体现出来。

一旦在 worker 进行计算，它对于用户来说是无缝地且不会影响到用户体验。

三、兼容性

四、基本概念

1.首先记得去判断是否支持

```
if (window.worker) { ...}
```

2.创建一个新的 worker 很简单

```
const myworker = new worker('worker.js');
```

postmessage() 方法和 onmessage 事件处理函数是 workers 的黑手机市场调查问卷魔法。

3. postmessage 用来发送消息，而 onmessage 用来监听消息

```
const worker = new worker('src/worker.js');worker.onmessage = e => { console.log(e.
data);};worker.postmessage('你好吗!');
```

在主线程中使用时， onmessage 和 postmessage() 必须挂在 worker 对象上，而在 worker 中使用时不用这样做。原因是，在 worker 内部， worker 是有效的全局作用域。

4.异常处理：

```
worker.onerror = function(error) { console.log(error.message); throw error;};
```

5.终止 worker

```
worker.terminate();
```

worker 线程会被立即杀死，不会有任何机会让它完成自己的操作或清理工作。

6.在 worker 线程中， workers 也可以调用自己的 cl工行网银助手下载ose 方法进行关闭：

close();

五、快速开始

为了快速掌握，我们来做一个小例子：项目结构如下

├——— index.html └——— src ├——— main.js └——— worker.js

html

<html><head> <title>web work demo</title> <meta charset="utf-8" /></head><body>
 hello jartto! <script src="src/main.js"></script></body></html
>

main.js

const worker = new worker('src/worker.js');worker.onmessage = e => { const messag
e = e.data; console.log(`[from worker]: ${message}`); document.getelementbyid('app').inn
erhtml = message;};worker.postmessage('写的真好!');

work.js

onmessage = e => { const message = e.data; console.log(`[from main]: ${messag
e}`); if(message.indexof('好') > -1) { postmessage('谢谢支持'); }};

代码很简单，主线程发送：「写的真好！」

web worker 收到消息，发现内容中含有「好」字，回传给主线程：「谢谢支持」

六、局限性

1.在 worker 内，不能直接操作 dom 节点，也不能使用 window
对象的默认方法和属性。然而我们可以使用大量 window 对象之下的东西，包括 websockets，
indexeddb 以及 firefox os 专用的 data store api 等数据存储机制。

这里举个例子，我们修改 main.js ：

const worker = new worker('src/worker.js');worker.onmessage = e => { const messag
e = e.data; console.log(`[from worker]: ${message}`); document.getelementbyid('app').inn
erhtml = message;};+ worker.onerror = function(error) {+ console.log(error);+ w
orker.terminate();+ };worker.postmessage('写的真好!');

再来修改 work.js

```
+ alert('jartto');onmessage = e => {  const message = e.data;  console.log(`[from
main]: ${message}`);  if(message.indexof('好') > -1) {    postmessage('谢谢支持');
                                                                    }};
```

这时候运行就会报出：

这是因为： worker.js 执行的上下文，与主页面 html
执行时的上下文并不相同，最顶层的对象并不是 window， woker.js 执行的全局上下文，而是 workerglobalscope，我们具体说明。

2. workers 和主线程间的数据传递通过这样的消息机制进行：双方都使用 postmessage() 方法发送各自的消息，使用 onmessage 事件处理函数来响应消息（消息被包含在 message 事件的 data 属性中）。

这个过程中数据并不是被共享而是被复制。

3.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。

4.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://），旅游小知识它所加载的脚本，必须来自服务器。

5.不允许本地文件

uncaught securityerror: failed to create a worker:
script at ‘(path)/worker.js’
cannot be accessed from origin ‘null’.

chrome doesn’t let you load web workers when running scripts from a local file.

那如何解决呢？我们可以启动一个本地服务器，建议使用 http-server，简单易用。

6.内容安全策略

有别于创建它的 document 对象， worker 有它自己的执行上下文。因此普遍来说， worker 并不受限于创建它的 document （或者父级 worker ）的内容安全策略。

我们来举个例子，假设一个 document 有如下头部声明：

content-security-policy: script-src 'self'

这个声明有一部分作用在于，禁止它内部包含的脚本代码使用 eval()
方法。然而，如果脚本代码创建了一个 worker，在 worker 上下文中执行的代码却是可以使用 eval() 的。

为了给 worker 指定 csp，必须为发送 worker 代码的请求本身加上一个 csp。

有一个例外情况，即 worker 脚本的源如果是一个全局性的唯一的标识符（例如，它的 url 指定了数据模式或者 blob），worker 则会继承创建它的 document 或者 worker 的 csp。

七、扩展：workerglobalscope

关于，我们可以在 mdn 上面找到文档：

1. self ：

我们可以使用 workerglobalscope 的 self 属性来获取这个对象本身的引用。

2. location ：

location 属性返回当线程被创建出来的时候与之关联的 workerlocation
对象，它表示用于初始化这个工作线程的脚步资源的绝对 url，即使页面被多次重定向后，这个 url 资源位置也不会改变。

3. close ：

关闭当前线程，四海山与 terminate 作用类似。

4. caches ：

当前上下文得 cachestorage，确保离线可用，同时可以自定义请求的响应。

5. console ：

支持 console 语法。

6. importscripts
我们可以通过 importscripts() 方法通过 url 在 worker 中加载库函数。

7. xmlhttprequest
有了它，才能发出 ajax 请求。

8.可以使用：

settimeout/setintervaladdeventlistener/postmessage

还有很多 api 可以使用，这里就不一一举例了。

八、异常处理

当 worker 出现运行中错误时，它的 onerror 事件处理函数会被调用。它会收到一个扩展了 errorevent 接口的名为 error 的事件。该事件不会冒泡并且可以被取消。

为了防止触发默认动作，worker 可以调用错误事件的 preventdefault() 方法。

错误事件我们常用如下这三个关键信息：

message：可读性良好的错误消息；filename：发生错误的脚本文件名；lineno：发生错误时所在脚本文件的行号；

```
worker.onerror = function(error) {  console.log(error.message);  throw error;};
```

以上就是本文的全部内容，希望对大家的学习有所帮助，也希望大家多多支持www.887551.com。