[**HTML5 Web Worker的使用**](https://www.cnblogs.com/520yang/articles/4763937.html)

Web Workers 是 HTML5 提供的一个javascript多线程解决方案，我们可以将一些大计算量的代码交由web Worker运行而不冻结用户界面。

**一：如何使用Worker**

Web Worker的基本原理就是在当前javascript的主线程中，使用Worker类加载一个javascript文件来开辟一个新的线程，起到互不阻塞执行的效果，并且提供主线程和新线程之间数据交换的接口：postMessage，onmessage。

那么如何使用呢，我们看一个例子：

//worker.js  
onmessage =function (evt){  
 var d = evt.data;//通过evt.data获得发送来的数据  
 postMessage( d );//将获取到的数据发送会主线程  
}

HTML页面：test.html

[复制代码](javascript:void(0);)

复制代码

<!DOCTYPE HTML>  
<html>  
<head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>  
 <script type="text/javascript">  
//WEB页主线程  
var worker =new Worker("worker.js"); //创建一个Worker对象并向它传递将在新线程中执行的脚本的URL  
 worker.postMessage("hello world"); //向worker发送数据  
 worker.onmessage =function(evt){ //接收worker传过来的数据函数  
 console.log(evt.data); //输出worker发送来的数据  
 }  
 </script>  
 </head>  
 <body></body>  
</html>

复制代码

[复制代码](javascript:void(0);)

　　用Chrome浏览器打开test.html后，控制台输出  "hello world" 表示程序执行成功。

通过这个例子我们可以看出使用web worker主要分为以下几部分

WEB主线程:

1.通过 worker = new Worker( url ) 加载一个JS文件来创建一个worker，同时返回一个worker实例。

2.通过worker.postMessage( data ) 方法来向worker发送数据。

3.绑定worker.onmessage方法来接收worker发送过来的数据。

4.可以使用 worker.terminate() 来终止一个worker的执行。

worker新线程：

1.通过postMessage( data ) 方法来向主线程发送数据。

2.绑定onmessage方法来接收主线程发送过来的数据。

**二：Worker能做什么**

知道了如何使用web worker ，那么它到底有什么用，可以帮我们解决那些问题呢。我们来看一个fibonacci数列的例子。

大家知道在数学上，fibonacci数列被以递归的方法定义：F0=0，F1=1，Fn=F(n-1)+F(n-2)（n>=2，n∈N\*），而javascript的常用实现为：

var fibonacci =function(n) {  
 return n <2? n : arguments.callee(n -1) + arguments.callee(n -2);  
};  
//fibonacci(36)

在chrome中用该方法进行39的fibonacci数列执行时间为19097毫秒 ，而要计算40的时候浏览器直接提示脚本忙了。

由于javascript是单线程执行的，在求数列的过程中浏览器不能执行其它javascript脚本，UI渲染线程也会被挂起，从而导致浏览器进入僵死状态。使用web worker将数列的计算过程放入一个新线程里去执行将避免这种情况的出现。具体看例子：

[复制代码](javascript:void(0);)

复制代码

//fibonacci.js  
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));  
};

复制代码

[复制代码](javascript:void(0);)

HTML页面：fibonacci.html

[复制代码](javascript:void(0);)

复制代码

<!DOCTYPE HTML>  
<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>  
<title>web worker fibonacci</title>  
<script type="text/javascript">  
 onload =function(){  
 var worker =new Worker('fibonacci.js');   
 worker.addEventListener('message', function(event) {  
 var timer2 = (new Date()).valueOf();  
 console.log( '结果：'+event.data, '时间:'+ timer2, '用时：'+ ( timer2 - timer ) );  
 }, false);  
 var timer = (new Date()).valueOf();  
 console.log('开始计算：40','时间:'+ timer );  
 setTimeout(function(){  
 console.log('定时器函数在计算数列时执行了', '时间:'+ (new Date()).valueOf() );  
 },1000);  
 worker.postMessage(40);  
 console.log('我在计算数列的时候执行了', '时间:'+ (new Date()).valueOf() );  
 }   
 </script>  
</head>  
<body>  
</body>  
</html>

复制代码

[复制代码](javascript:void(0);)

在Chrome中打开fibonacci.html，控制台得到如下输出：

开始计算：40 时间:1316508212705  
我在计算数列的时候执行了 时间:1316508212734  
定时器函数在计算数列时执行了 时间:1316508213735  
结果：102334155 时间:1316508262820 用时：50115

这个例子说明在worker中执行的fibonacci数列的计算并不会影响到主线程的代码执行，完全在自己独立的线程中计算，只是在计算完成之后将结果发回主线程。

利用web worker我们可以在前端执行一些复杂的大量运算而不会影响页面的展示，并且不会弹出恶心的脚本正忙提示。

下面这个例子使用了web worker来计算场景中的像素，场景打开时是一片一片进行绘制的，一个worker只计算一块像素值。

<http://nerget.com/rayjs-mt/rayjs.html>

**三：Worker的其他尝试**  
我们已经知道Worker通过接收一个URL来创建一个worker，那么我们是否可以利用web worker来做一些类似jsonp的请求呢，大家知道jsonp是通过插入script标签来加载json数据的，而script元素在加载和执行过程中都是阻塞式的，如果能利用web worker实现异步加载将会非常不错。

下面这个例子将通过 web worker、jsonp、ajax三种不同的方式来加载一个169.42KB大小的JSON数据

[复制代码](javascript:void(0);)

复制代码

// /aj/webWorker/core.js  
function $E(id) {  
 return document.getElementById(id);  
}  
onload =function() {  
 //通过web worker加载  
 $E('workerLoad').onclick =function() {  
 var url ='http://js.wcdn.cn/aj/mblog/face2';  
 var d = (new Date()).valueOf();  
 var worker =new Worker(url);  
 worker.onmessage =function(obj) {  
 console.log('web worker: '+ ((new Date()).valueOf() - d));  
 };  
 };  
 //通过jsonp加载  
 $E('jsonpLoad').onclick =function() {  
 var url ='http://js.wcdn.cn/aj/mblog/face1';  
 var d = (new Date()).valueOf();  
 STK.core.io.scriptLoader({  
 method:'post',  
 url : url,  
 onComplete : function() {  
 console.log('jsonp: '+ ((new Date()).valueOf() - d));  
 }  
 });  
 };  
 //通过ajax加载  
 $E('ajaxLoad').onclick =function() {  
 var url ='http://js.wcdn.cn/aj/mblog/face';  
 var d = (new Date()).valueOf();  
 STK.core.io.ajax({  
 url : url,  
 onComplete : function(json) {  
 console.log('ajax: '+ ((new Date()).valueOf() - d));  
 }  
 });  
 };  
};

复制代码

[复制代码](javascript:void(0);)

HTML页面：/aj/webWorker/worker.html

[复制代码](javascript:void(0);)

复制代码

<!DOCTYPE HTML>  
<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>  
<title>Worker example: load data</title>  
<script src="http://js.t.sinajs.cn/STK/js/gaea.1.14.js" type="text/javascript"></script>  
<script type="text/javascript" src="http://js.wcdn.cn/aj/webWorker/core.js"></script>  
</head>  
<body>  
 <input type="button" id="workerLoad" value="web worker加载"></input>  
 <input type="button" id="jsonpLoad" value="jsonp加载"></input>  
 <input type="button" id="ajaxLoad" value="ajax加载"></input>  
</body>  
</html>

复制代码

[复制代码](javascript:void(0);)

设置HOST

127.0.0.1 js.wcdn.cn

通过 <http://js.wcdn.cn/aj/webWorker/worker.html> 访问页面然后分别通过三种方式加载数据，得到控制台输出：

web worker: 174  
jsonp: 25  
ajax: 38

多试几次发现通过jsonp和ajax加载数据的时间相差不大，而web worker的加载时间一直处于高位，所以用web worker来加载数据还是比较慢的，即便是大数据量情况下也没任何优势，可能是Worker初始化新起线程比较耗时间。除了在加载过程中是无阻塞的之外没有任何优势。

那么web worker是否能支持跨域js加载呢，这次我们通过<http://127.0.0.1/aj/webWorker/worker.html> 来访问页面，当点击 "web worker加载" 加载按钮时Chrome下无任何反映，FF6下提示错误。由此我们可以知道web worker是不支持跨域加载JS的，这对于将静态文件部署到单独的静态服务器的网站来说是个坏消息。

所以web worker只能用来加载同域下的json数据，而这方面ajax已经可以做到了，而且效率更高更通用。还是让Worker做它自己擅长的事吧。

**四：总结**

web worker看起来很美好，但处处是魔鬼。

我们可以做什么：

1.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

2.可以在worker中通过importScripts(url)加载另外的脚本文件

3.可以使用 setTimeout(), clearTimeout(), setInterval(), and clearInterval()

4.可以使用XMLHttpRequest来发送请求

5.可以访问navigator的部分属性

有那些局限性：

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

4.不是每个浏览器都支持这个新特性