HTML5 Web Worker实践报告

###### **Web Worker理解**

工作线程（Web Worker）是HTML5 中提出的概念，并且规范出 Web Worker 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。Web Worker 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。 即为开辟新线程，充分利用计算机算力，防止“假死”（单线程，js引擎和UI线程互斥），所以Web Worker 的最佳使用场景是执行一些开销较大的数据处理或计算任务。另一方面，Web workers 承担了所有繁重的任务，因此不存在弹出脚本运行缓慢警告的风险，也不需要手动地将任务分割成多份执行（没有Web Workers 必须考虑）。

应用场景

　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

限制：

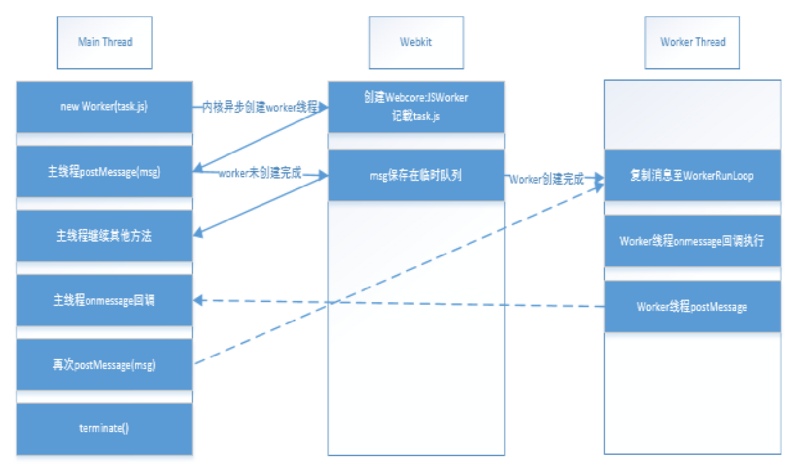
　1.不能跨域加载JS

　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.浏览器对WebSocket支持性不一致

运行原理：



###### **Web Worker 使用**

**应用形式：**

1. 主页面 主线程

2. webWorker.js

**常用API：**

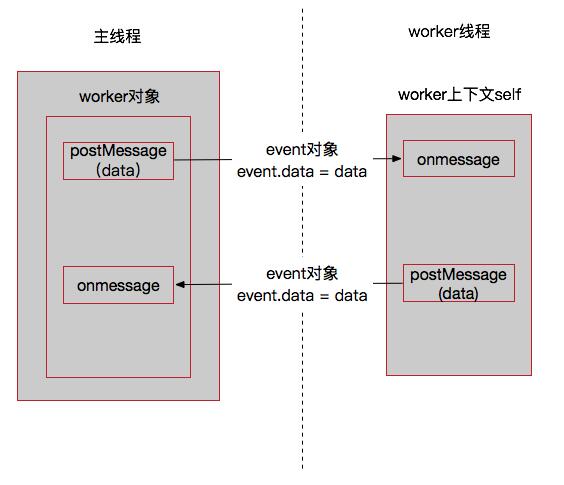
1. postMessage(data) 子线程与主线程之间互相通信使用方法，传递data

2. terminate() 主线程中终止worker，此后无法在利用其进行消息传递。一旦terminate后，无法重新启用，只能另外创建。

3. message 当有消息发送时触发该事件。双向传递消息，消息内容通过data来获取。

4. error 错误处理。错误消息通过e.message获取

**总结Web Worker 使用**：在“主线程”中执行如下操作即可创建一个 Worker 实例，“主线程”和Worker 之间通过 postMessage 发送消息，通过监听 onmessage 事件来接收消息，从而实现二者的通信。



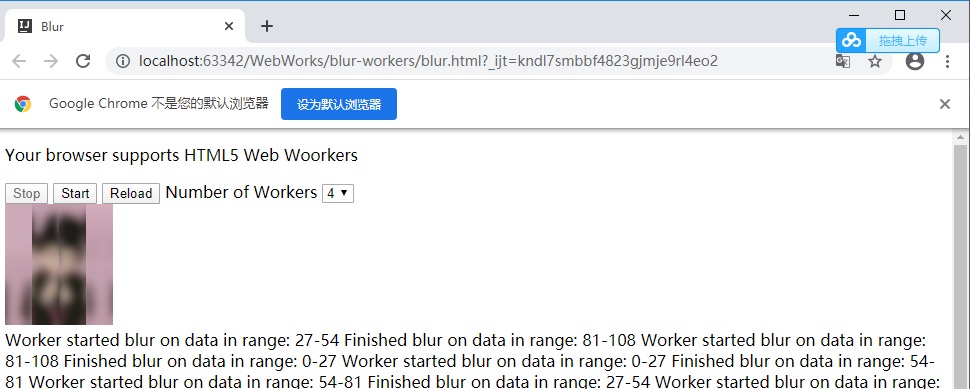
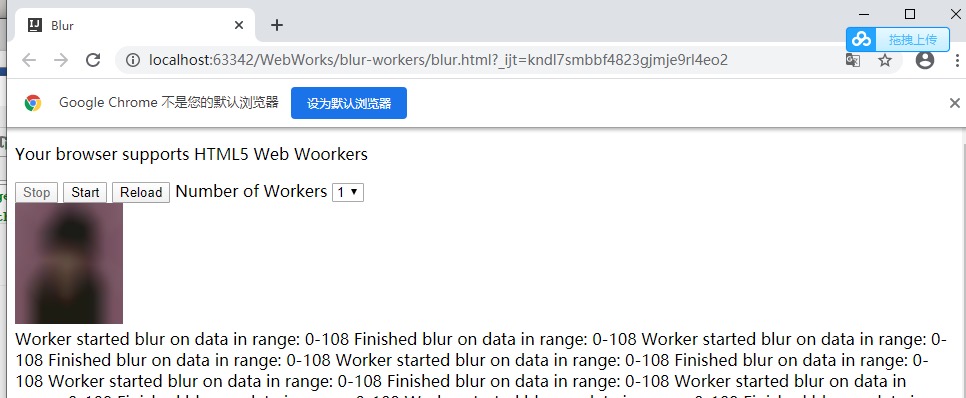
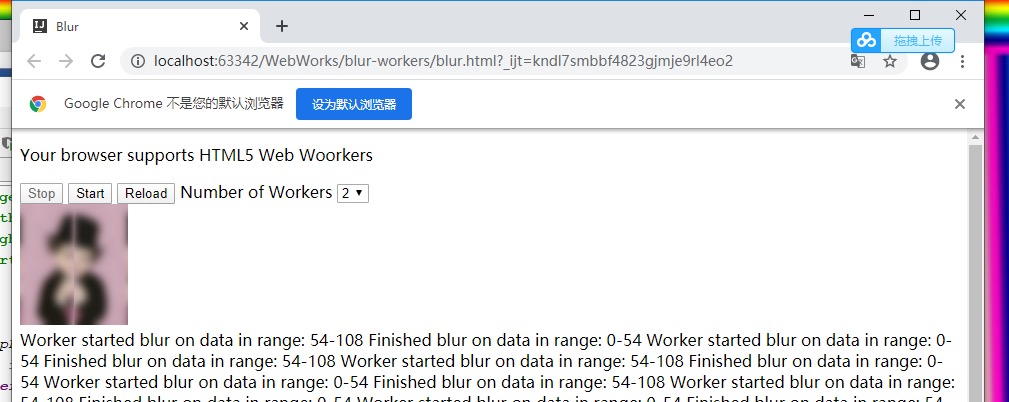
#### 实践说明

#### blur-workers （代码见blur-workers下）

**（完整实现后台复杂计算的应用场景 计算模糊像素点位置）**

页面实现对一张图片的淡化处理通过选择不同数量的worker同时对一张图片的不同区域淡化。

效果展示：



**实验发现**注意点：

**一．Terminate的细节**

worker线程从上到下同步运行它的代码，然后进入异步阶段来对事件及计时器响应，如果worker注册了message事件处理程序，只要其有可能触发，worker就一直在内存中，不会退出，所以通信完毕后得手动在主线程中terminate或者子线程中close掉，但如果worker没有监听消息，那么所有任务执行完毕（包括计数器）后就会退出。

（这个现象在实验中blur中碰见过，当所有像素点进行blur任务完成后，未中断时仍会一直打印message信息直至调用stop终止线程）

二．**Worker作用域**

当创建一个新的worker时，该代码会运行在一个全新的javascript的环境中（WorkerGlobalScope）运行,是完全和创建worker的脚本隔离，创建新worker的脚本叫做主线程，而被创建的新的worker叫做子线程。因此worker.js执行的上下文，与主页面html执行的上下文并不相同，最顶层的对象并不是window，worker.js执行的全局上下文WorkGlobalScope，所以无法访问window，与window相关的DOM API，但是可以与setTimeout，setInterval等协作。

经查找WorkerGlobalScope作用域下的常用属性：

1. self

可以使用GlobalScope的self属性来对这个对象本身引用

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，他表示用于初始化这个工作线程的资源的绝对URL，即使这个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，与terminate作用类似

4. importScripts

通过这个方法使用url在worker中加载库函数

5. XMLHttpRequest

支持Ajax异步请求

6. SetTimeout.setInterval以及addEventListener/postMessage

##### 2.shared worker 及worker（代码见shared-worker下）

该线程分为两种：dedicated worker和shared worker；dedicated worker只能被初始化它的js上下文中使用；shared worker可以在多个js上下文中使用。通常使用的worker是dedicated worker，它的工作情况可以通过chrome的调试工具查看。

new worker( )：专属线程，只针对当前运行环境

new sharedWorker( )：共享线程，同源策略下，多个运行环境共用同一个线程，包括数据。

##### 3. 优化woker与主线程通信开销（代码见optimizeMessageTransfer下）

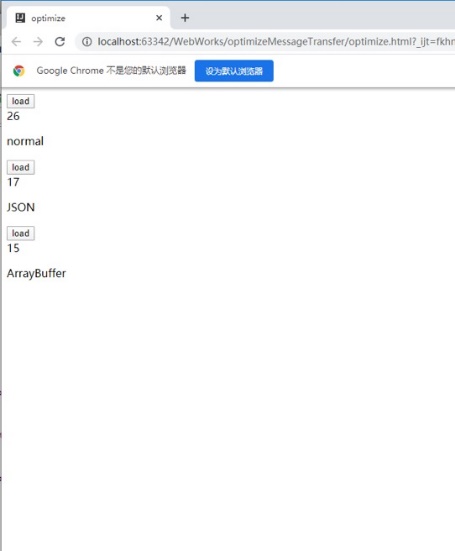
Worker 与“主线程”之间的数据传递默认是通过结构化克隆（Structured Clone）完成的。数据量较大时，克隆过程会比较耗时，这会影响 postMessage 和 onmessage 函数的执行时间。

尝试一：

通过 JSON.stringify 将对象序列化，接收之后再用 JSON.parse 还原。因为：stringfiy + 传递字符串的耗时 < 传递对象的耗时 。

尝试二：

避开克隆传值的方法，使用Transferable Objects，主要是采用二进制的存储方式，采用地址引用，解决数据交换的实时性问题；Transferable Objects支持的常用数据类型有ArrayBuffer和ImageBitmap；（注：可能时间太短了，单位毫秒优化结果并不稳定）



### 【注意】线程间的通讯是传值而不是传地址

主线程与子线程数据通信方式有多种，通信内容，可以是文本，也可以是对象。需要注意的是，这种通信是拷贝关系，即是传值而不是地址，子线程对通信内容的修改，不会影响到主线程。事实上，浏览器内部的运行机制是，先将通信内容串行化，然后把串行化后的字符串发给子线程，后者再将它还原。

参考资料：

1.MDN web docs :https://developer.mozilla.org/zh-CN/docs/Web/API/Worker

2.web worker是什么？理解并使用web worker：http://www.fly63.com/article/detial/998

3. 《HTML5高级程序设计》