JS是单线程的，JS在执行的时候是无法去响应用户的输入的，如果JS的主线程去执行一个计算密集型的任务，那么它势必会造成页面解析的延迟、页面的冻结。

Web worker就是解决这一问题的，而且web worker是处在一个自包含的执行环境中，它是无法去访问DOM的，这样也就避免了异步修改DOM。

API?

主线程中，

创建：Var worker=new Worker(“loader.js”); 必须同源

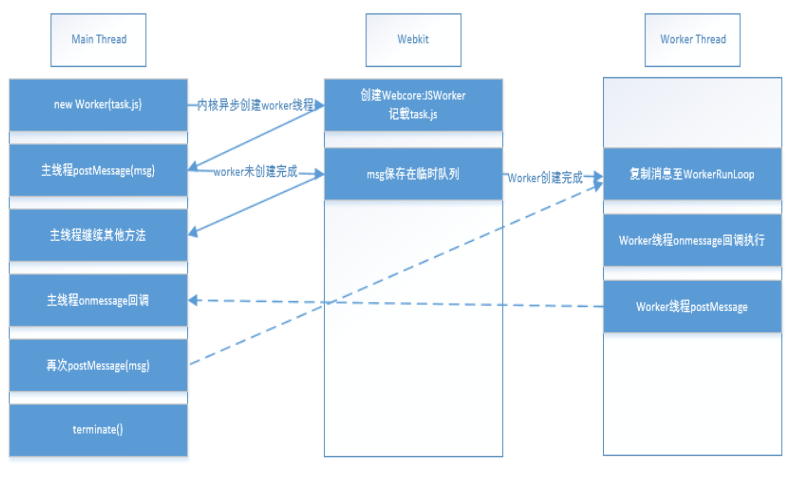
传递消息：worker.postMessage（”{“a”:1}”）；

监听消息:worker.onmessage =function(e){} e.data

监听错误：worker.onerror

停止worker:worker.terminate()

Worker线程中，onmessage和postmessage都是全局函数，可以直接访问



1. worker线程的创建是异步的  
   执行到new worker的时候，会异步创建worker线程，主线程的代码不会堵塞在这里等待worker去加载指定、执行指定的脚本文件，而是会继续向下执行后面的代码。
2. postmessage消息交互是由内核调度的  
   创建worker线程后，立即调用了postmessage传递数据。但是其实，在worker线程还没创建完成的时候，主线程发出的message会被临时存储在一个消息队列中，当异步创建worker线程完毕之后，临时消息队列会将数据结构性复制到worker中，worker开始处理消息。
3. worker线程之间数据通讯的方式  
   是结构性复制，当然也可以传输二进制数据，比如FILE、ArrayBuffer等对象。但是遂于大数据，会复制会造成性能问题。因此，JS允许主线程把二进制数据直接转移给子线程，但是转移后主线程就无法在使用这些数据了。这是为了防止多个线程同时修改数据的问题，这种数据转移的方法叫做Transferable Objects。  
     
     
   认识：  
   web worker带来了JS的多线程，我觉得是不对的。你认为是多线程，那么就要想到设计web worker的初衷，他就是为了利用现代计算机多核处理器，利用他的计算能力，初衷不是多线程，如果真是为了多线程，那么web worker怎么会设计到不允许它操作DOM呢？  
   我认为web worker的初衷就是为了带来以一种后台的计算能力，能够防止计算密集型的任务老是在主线程中，这样会导致页面解析的延迟，UI的冻结，它实际上是更高一层的“异步”。  
   应用场景？  
   1.使用专用的线程进行数学运算  
   2.图像处理---虽然无法postmessage图片的DOM元素，但是我们可以利用canvas获得一个ImageData对象传递，让worker线程去处理  
   3.大量数据的检索---当你需要在调用ajax后处理大量数据，那么可以在web worker中去做这些，这样可以避免页面冻结。