Web Worker 在 WebGL 地图引擎中的实践|第一期

J 百度地图技术团队 2017-04-20

百度地图技术团队认真服务每一位技术爱好者，不定期更新原创技术架构分享；推出技术沙龙、技术公开课等活动！

Web Worker 扫盲

Web Worker 对于前端开发人员来说一定不陌生，即使没有在实际项目中使用过，那么也一定有所了解。Web Worker 是为了解决 JavaScript 在浏览器环境中没有多线程的问题。支持 Web Worker 的浏览器会额外提供一个 JavaScript Runtime 供 Web Worker 使用。需要注意的是在 Web Worker 中不能使用和访问 DOM 元素，也访问不到 Window 对象。它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。

Web Worker 使用起来非常简单，在“主线程”中执行如下操作即可创建一个 Worker 实例，通过监听 onmessage 事件获取消息，通过 postMessage 发送消息：

```
var worker = new Worker('worker.js');
worker.onmessage = function (e) {
    var data = e.data;
}
worker.postMessage(['message']);
```

worker.js 中的代码如下：

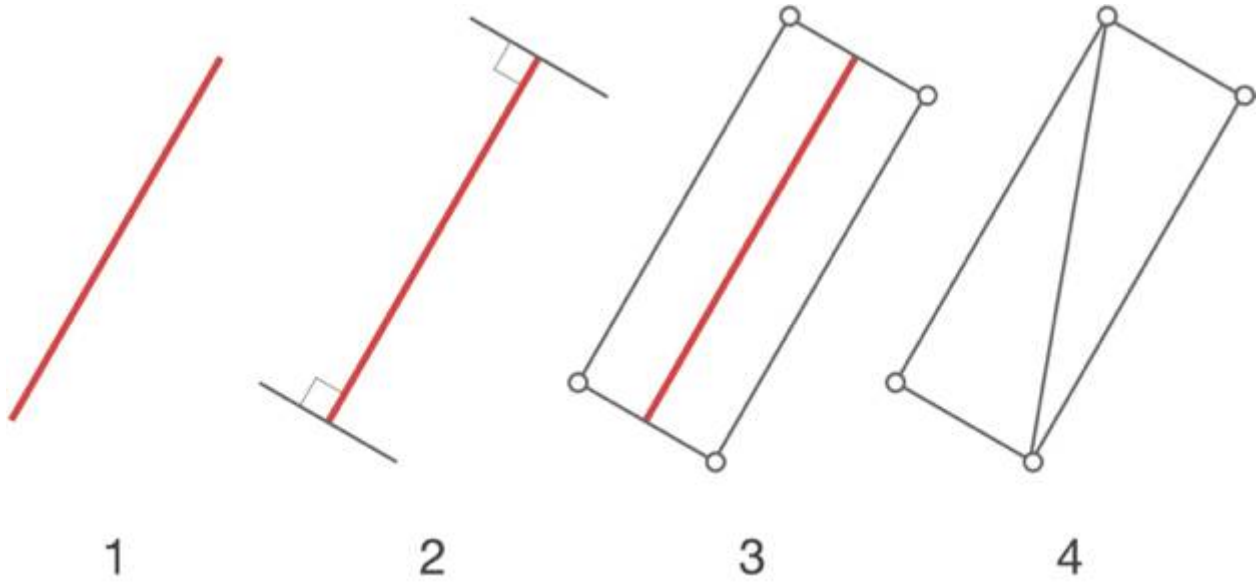
```
self.onmessage = function(e) {
    var messages = e.data;
    var workerResult = {};
    // do something
    ...
    postMessage(workerResult);
}
```

“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。

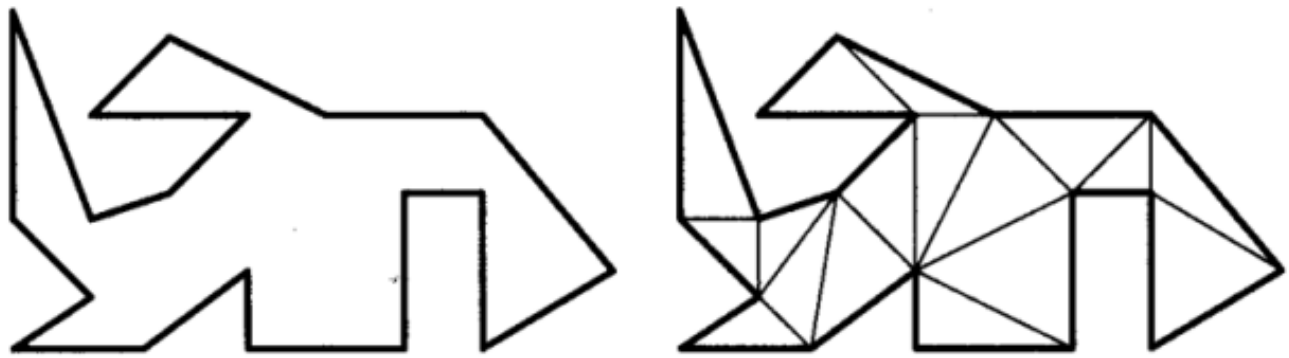
为什么使用 Web Worker

在对 Web Worker 进行了简单扫盲之后，接下来说说为什么要在 WebGL 地图引擎中使用它。熟悉 WebGL 或者 OpenGL 的会知道，不论多么复杂的图形或模型，最终都是由若干三角形组成，地图也不例外。地图上表示绿地、水系的面以及各种道路最终都需要通过三角形来描述，所以在绘制之前我们需要对原始数据进行如下加工：

线的处理过程示意：



面的处理过程示意：



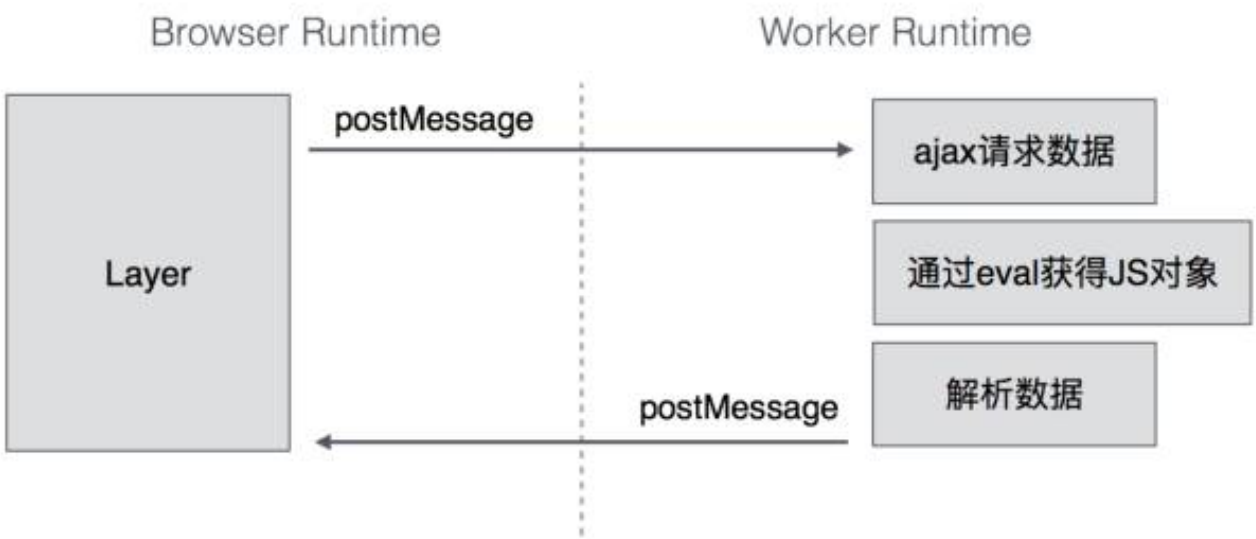
这个过程称为**三角形剖分（Triangulation）**。

通常一个地图网格中会包含上千个坐标点，对一个网格的数据处理通常会消耗几十毫秒甚至到百毫秒，一个1280x800的屏幕会使用20到30个网格，那么整个屏幕的数据处理时间就要消耗几百毫秒甚至到1、2秒。虽然从时间长度来看1、2秒的消耗不算太多，但是这1、2秒如果全部在“主线程”执行的话，浏览器在执行过程中无法响应其他操作，就会造成卡顿。即使通过一些优化手段可以将每个网格的解析过程拆分开，那么处理一个网格所需的几十毫秒同样会影响页面的操作流畅性。因为以60FPS为目标的话，每一帧的处理时间不

能超过16.6ms（即 1s / 60。在 Google 提出的 RAIL 模型中，建议不超过10ms）。数据处理过程是纯数学计算，不需要使用DOM，也不需要访问Window对象，非常适合用 Web Worker 来处理。

另外一点，前端人员都知道用 JSONP 来加载异步数据，没错 JSONP 的加载过程确实是异步的，但是当数据返回时，它本质上是脚本的执行，通过调用一个全局方法把数据赋值给某个对象。由于网格数据量较大，在使用 JSONP 方式获取网格数据时发现脚本执行的过程竟然也有十几毫秒的开销，这同样会影响页面流畅性。幸好在 Web Worker 中可以使用 XMLHttpRequest 对象，这样可以直接在 Worker “线程”发起请求，请求结果直接在 Worker 中处理，这样也减少了一次“主线程”和 Worker 的大数据量通信（后面会讲到，“主线程”和 Web Worker 之间的大数据量通信也有一定的性能开销）。

最终方案如下所示：



- “主线程”的模块发送消息给 Worker 告知其加载数据的url。
- Worker 通过 ajax 发送请求。
- 返回的数据通过 eval 方式得到对象进而交给解析器解析。
- 解析结束之后将数据传给“主线程”。

使用多少个 Web Worker

确定要使用 Web Worker 之后，确定多少个 Worker 同时工作就是下面需要考虑的问题，数量少了发挥不出并行处理的优势，数量多了有可能导致 Worker 处理速度变慢。

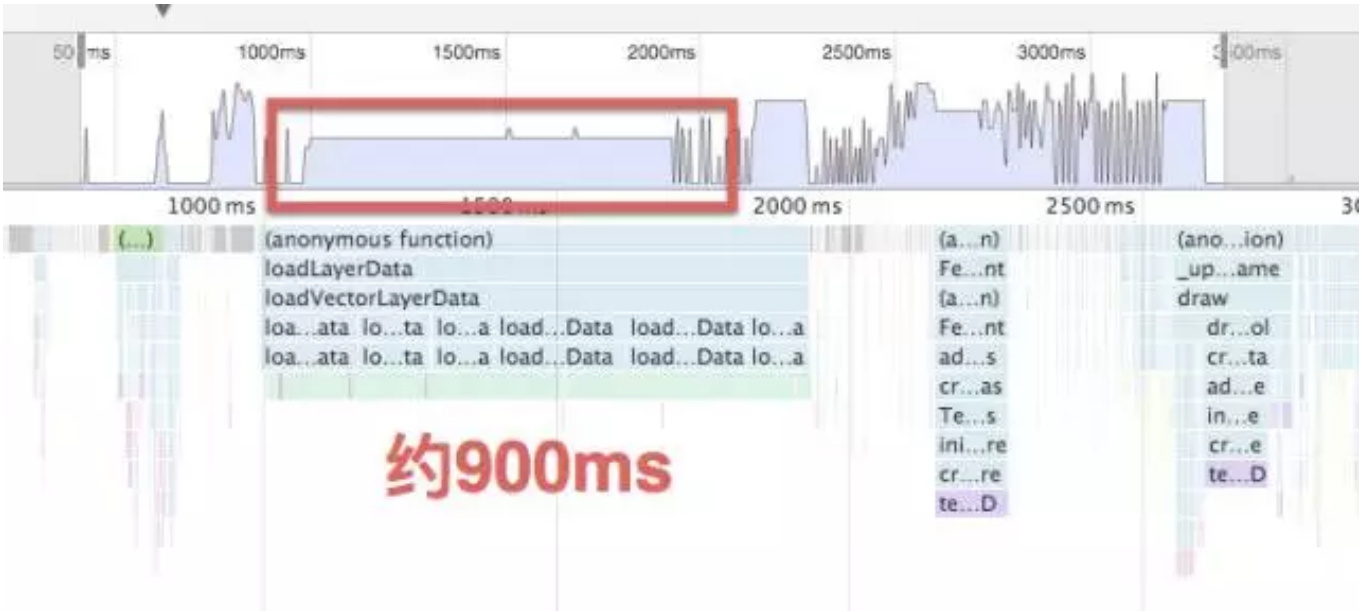
一般的做法是读取 navigator.hardwareConcurrency 这个属性，它表示机器最多可并行执行的任务数量，如果取不到这个值，可以给一个默认值，例如4。还有一种动态检测 Worker 数量的方法，有兴趣的话可以看：https://github.com/oftn-oswg/core-estimator （可长按复制在浏览器中打开）。

有人可能会问，假设一个机器的最大并行数是8，那么是不是只能创建7个 Worker，留一个给“主线程”使用？我的建议是要看的你的应用的实际情况，通常来说页面的“主线程”不会长时间执行操作，大部分时间都处于空闲状态，那么这时候 Worker 数量完全可以取8。通过 Chrome Dev Tools 的 Timeline（新版叫做 Performance）工具可以查看每个 Worker 的工作情况，确定是否影响“主线程”工作。

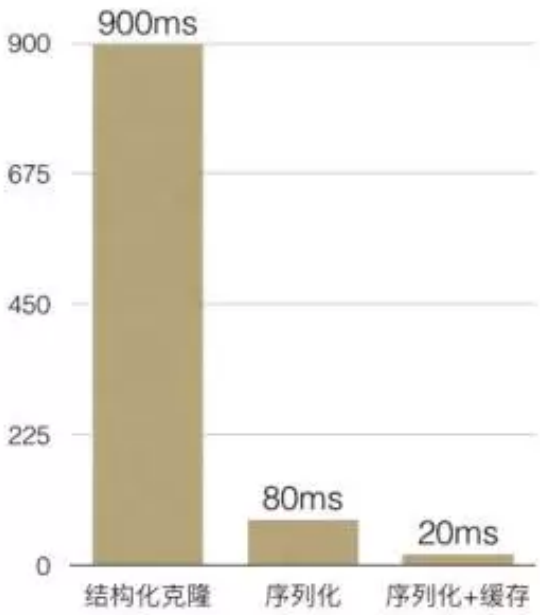
Web Worker 与“主线程”通信的开销

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。

在地图引擎中处理数据之前，“主线程”需要将一个样式描述文件传给 Worker，最开始这个过程总共耗时900ms（8个 Worker 的总耗时）：



解决的办法是先通过 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 。**



优化后时间由原来的900ms降低到80ms，由于序列化只需要执行一次，后续无需再次执行，缓存之后耗时降低到20ms。

当 Worker 处理完数据之后，需要把数据返回给主线程，在这个过程中耗时如下所示：

传递方法	worker 的 postMessage 耗时	“主线程” onmessage 耗时
对象传递	100ms~150ms	50ms~80ms
序列化	**20ms~50ms**	**10ms**

虽然使用序列化方案时间有所降低，但是“主线程”的10ms还是有些多。是否还有其他方法优化呢?

Web Worker 中有个概念叫做 Transferable Objects（暂且译为可传递对象），一旦对象是 Transferable 的，那么传递过程不再使用结构化克隆，而是按引用传递，这就避免了克隆导致的额外开销。当然不是所有的对象都可以，只有诸如 ArrayBuffer、ImageBitmap 这样的二进制数据类型才可以。

使用方式如下：

```
postMessage({
    road: roadData, // Float32Array
    area: areaData, // Float32Array
    other: someSmallData // normal Object
},
[roadData.buffer, areaData.buffer]);
```

postMessage 的第二个参数是一个 list，里面说明了哪些对象是可传递的。

对于 WebGL 来说，缓冲区所需要的数据类型恰好是各种类型数组（Typed Array），因此可以在 Worker 中将处理好的数据提前转换为类型数组，进而可以按引用方式传递。

最终优化结果如下：

传递方法	worker 的 postMessage 耗时	“主线程” onmessage 耗时
对象传递	100ms~150ms	50ms~80ms
序列化	20ms~50ms	10ms
可传递对象	**1ms**	**1ms**

可以看到不论是发送数据还是接收数据，时间都在1ms，收益可以说是 amazing!

如果你的项目中并不需要使用类型数组，也可以把数据先 pack 成为类型数组，收到之后再 unpack 还原，当然你需要对比两种方案的耗时各是多少来决定是否使用。

Web Worker 业界实践

前面提到的通过序列化和可传递对象方式提升Worker传递数据的性能在业界已经有广泛实践。上面提到的pack/unpack 方法在 WebGL Iǹšiǵẖţšḱ一书中也有提到。

在这里想提一个关于 WebGL 与 Web Worker 结合的方案：**WebGLWorker**。

WebGLWorker 是由 Mozilla 的工程师提出来的，核心思路是在 Worker 中提供一套 WebGL 方法，通过 postMessage 将 WebGL 命令传给“主线程”，“主线程”将命令添加到队列中，通过适当的调度，将命令转为真正的 WebGL 调用，从而达到提升性能的目的。有兴趣的同学可以参考这篇文章：https://research.mozilla.org/2014/07/22/webgl-in-web-workers-today-and-faster-than-expected/ （可长按复制在浏览器中打开）。

Web Worker 使用小结

- Web Worker 独立于浏览器“主线程”工作，但不能访问 DOM 和 Window 对象。
- Web Worker 可以使用 XMLHttpRequest 来发送数据请求。
- 根据 hardwareConcurrency 确定 Worker 数量，并实际测试效果。
- 数据传递考虑使用序列化或者可传递对象方式降低数据传递开销。

END

▼

ID:Map_technology

长按左侧二维码关注