

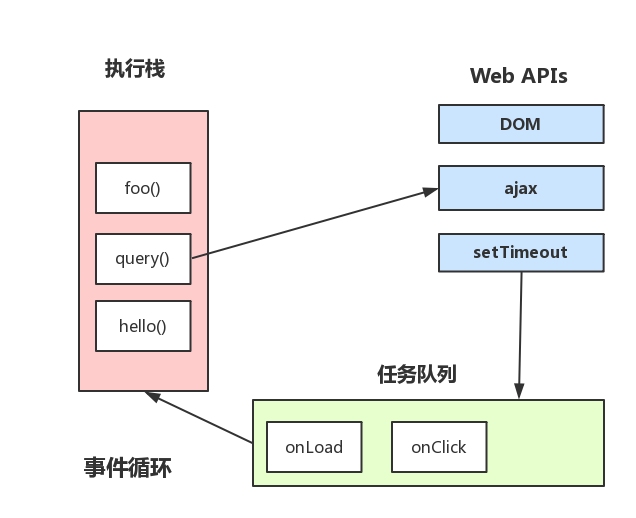
1. GUI渲染线程
   * 负责渲染浏览器界面，解析HTML，CSS，构建DOM树和Render树，布局和绘制等。
   * 当界面需要重绘（Repaint）或由于某种操作引发回流(reflow)时，该线程就会执行
   * 注意，**GUI渲染线程与JS引擎线程是互斥的**，当JS引擎执行时GUI线程会被挂起（相当于被冻结了），GUI更新会被保存在一个队列中**等到JS引擎空闲时**立即被执行。
2. JS引擎线程
   * 也称为JS内核，负责处理Javascript脚本程序。（例如V8引擎）
   * JS引擎线程负责解析Javascript脚本，运行代码。
   * JS引擎一直等待着任务队列中任务的到来，然后加以处理，一个Tab页（renderer进程）中无论什么时候都只有一个JS线程在运行JS程序
   * 同样注意，**GUI渲染线程与JS引擎线程是互斥的**，所以如果JS执行的时间过长，这样就会造成页面的渲染不连贯，导致页面渲染加载阻塞。
3. 事件触发线程
   * 归属于浏览器而不是JS引擎，用来控制事件循环（可以理解，JS引擎自己都忙不过来，需要浏览器另开线程协助）
   * 当JS引擎执行代码块如setTimeOut时（也可来自浏览器内核的其他线程,如鼠标点击、AJAX异步请求等），会将对应任务添加到事件线程中
   * 当对应的事件符合触发条件被触发时，该线程会把事件添加到待处理队列的队尾，等待JS引擎的处理
   * 注意，由于JS的单线程关系，所以这些待处理队列中的事件都得排队等待JS引擎处理（当JS引擎空闲时才会去执行）
4. 定时触发器线程
   * 传说中的setInterval与setTimeout所在线程
   * 浏览器定时计数器并不是由JavaScript引擎计数的,（因为JavaScript引擎是单线程的, 如果处于阻塞线程状态就会影响记计时的准确）
   * 因此通过单独线程来计时并触发定时（计时完毕后，添加到事件队列中，等待JS引擎空闲后执行）
   * 注意，W3C在HTML标准中规定，规定要求setTimeout中低于4ms的时间间隔算为4ms。
5. 异步http请求线程
   * 在XMLHttpRequest在连接后是通过浏览器新开一个线程请求
   * 将检测到状态变更时，如果设置有回调函数，异步线程就**产生状态变更事件**，将这个回调再放入事件队列中。再由JavaScript引擎执行。

从Event Loop谈JS的运行机制

* JS引擎线程
* 事件触发线程
* 定时触发器线程

理解一个概念：

* JS分为同步任务和异步任务
* 同步任务都在主线程上执行，形成一个执行栈
* 主线程之外，**事件触发线程**管理着一个任务队列，只要异步任务有了运行结果，就在任务队列之中放置一个事件。
* 一旦执行栈中的所有同步任务执行完毕（此时JS引擎空闲），系统就会读取任务队列，将可运行的异步任务添加到可执行栈中，开始执行。



微任务包括：process.nextTick（Nodejs）, Promises, Object.observe, MutationObserver;

任务包括：script（整体代码）, setTimeout, setInterval, setImmediate, I/O, UI rendering

原文链接：https://segmentfault.com/a/1190000012925872

动画演示：https://jakearchibald.com/2015/tasks-microtasks-queues-and-schedules/