JavaScript的并发模型基于`event loop`。

为什么JavaScript是单线程

> IO的开销是巨大的

当前变成技术中最大的浪费来自于等待I/O操作的完成。有几种方法可以解决性能的影响：
参见cnode社区文章

- 同步方式：按次序一个一个的处理请求。利：简单；弊：任何一个请求都可以阻塞其他所有请求。
- 开启新进程：每个请求都开启一个新进程。利：简单；弊：大量的链接意味着大量的进程。
- 开启新线程：每个请求都开启一个新线程。利：简单，而且跟进程比，对系统内核更加友好，因为线程比进程轻的多;弊:不是所有的机器都支持线程，而且对于要处理共享资源的情况，多线程编程会很快变得太过于复杂。

JavaScript是单线程，同一时间只能做一件事。

作为浏览器脚本语言，JavaScript主要用途是处理用户交互以及操作dom，决定了它只能是单线程，否则带来很复杂的同步问题。
比如，JavaScript同时又两个线程，一个在dom节点添加内容，另一个线程删除了这个节点。
HTML5提出的web worker标准，允许JavaScript脚本创建多个线程，但是子进程完全受主线程控制，且不得操作dom。

任务队列

单线程意味着所有任务需要**排队**，前一个任务结束才会执行后一个任务，如果前一个任务耗时很长，后一个任务不得不等待。
排队一般cpu是闲着的，io很慢导致的。
主线程可以完全不管IO设备，挂起处于等待中的任务，先运行排在后面的任务，等到IO设备返回了结果，再回过头把挂起的任务继续执行下去。

任务可以划分两种：

1. 同步任务（synchronous）:主线程上执行的任务，只有前一个任务执行完毕才能执行后一个任务。
2. 异步任务（asynchronous）：不进入主线程，而进入task queue任务，只有task queue通知主线程，某个异步任务可以执行了，该任务才会进入主线程执行。

执行机制

- 所有同步任务都在主线程上执行，形成一个执行栈（execution context stack）。
- 主线程之外，还存在一个任务队列，只要异步任务有了运行结果，就在任务队列中放置一个事件。
- 一旦执行栈中所有的同步任务执行完毕，系统就会读取任务队列，看看里面有哪些事件，对应的异步任务结束等待状态，进入执行栈开始执行。
- 主线程不断重复上面的第三步。

只要主线程空了就会去读取任务队列。这就是JavaScript的运行机制。

事件和回调函数

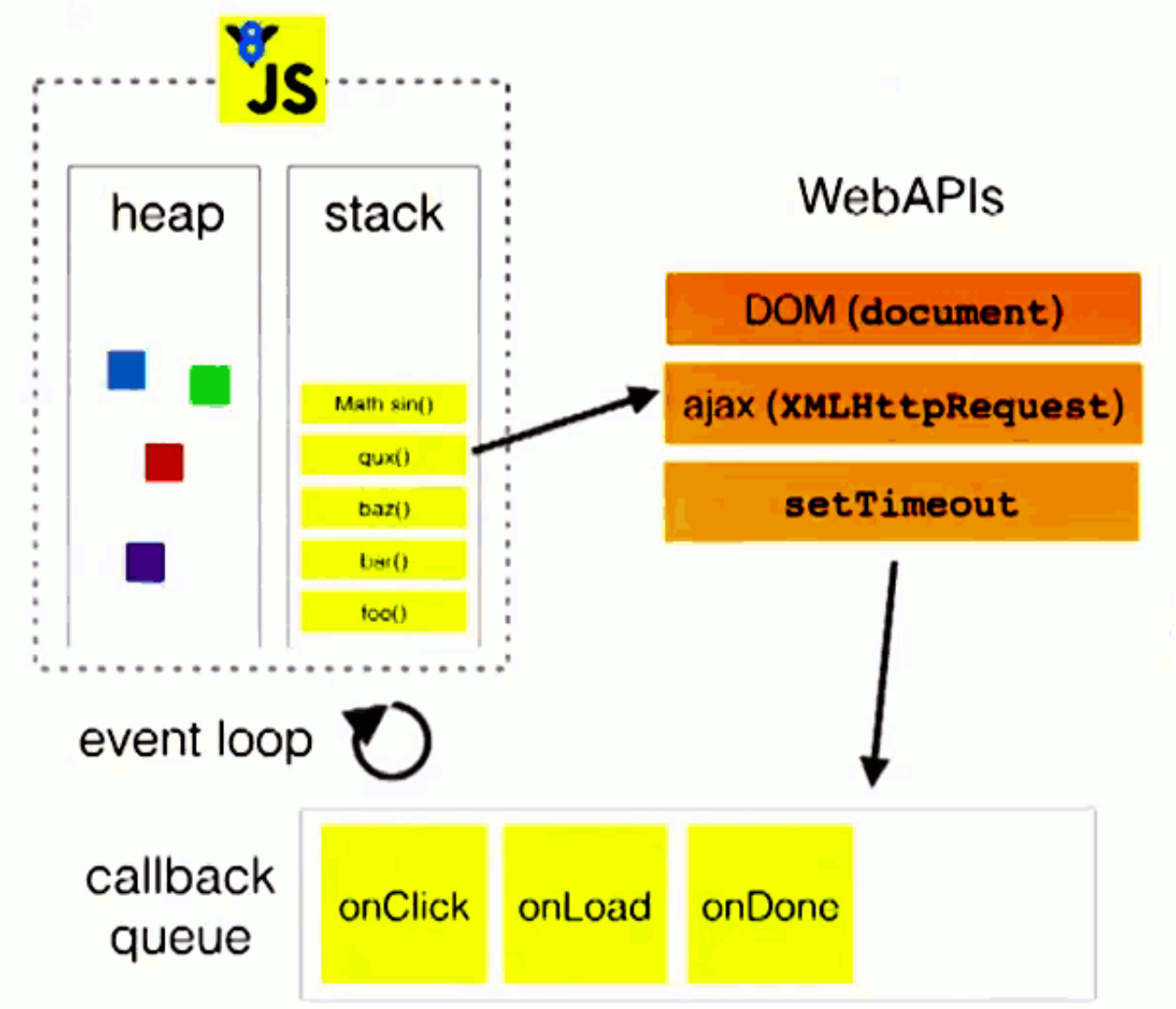
任务队列是一个事件的队列，也可以理解成消息的队列，IO设备完成一项任务就在任务队列中添加一个事件，表示相关的异步任务可以进入执行栈了，主线程读取任务队列，就是读取里面有哪些事件。

任务队列中的事件，除IO设备以外还包括用户产生的交互事件，只要指定过回调函数，这些事件发生时就会进入任务队列，等待主线程读取。

所谓回调函数callback就是那些会被主线程挂起来的代码，异步任务必须指定回调函数，当主线程开始执行异步任务，就是执行对应的回调函数。

Event Loop

主线程从任务队列中读取事件，这个过程是循环不断的，所以整个运行机制又称为Event Loop（事件循环）。



主线程运行时候，产生堆（heap）和（stack）,栈中的代码调用各种外部api，它们在任务队列中加入各种事件（click,load,done）。只要栈中的代码执行完毕，主线程就会去读取任务队列，依次执行那些事件所对应的回调函数。

执行栈中的代码（同步任务），总是在读取任务队列（异步任务）之前执行。

定时器

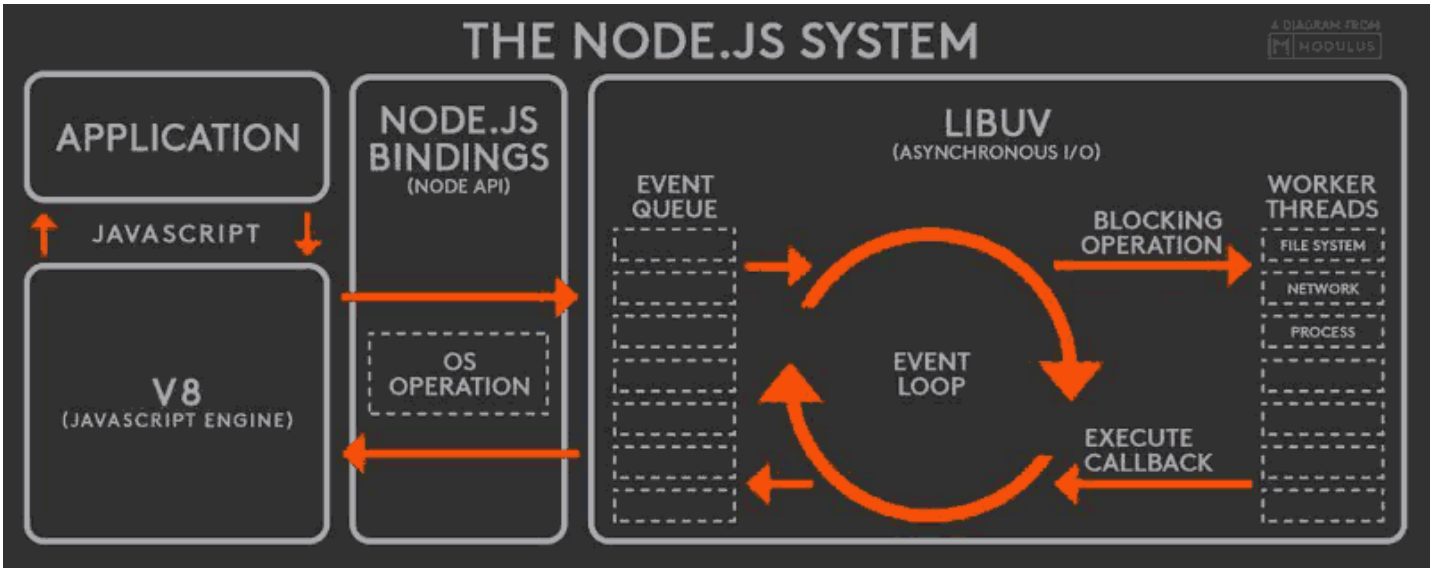
除了放置异步任务的事件，任务队列还可以放置定时事件，即指定代码在多少时间之后执行，这叫做timer的功能，也就是定时执行的代码。

setTimeout(fn,0)的含义是，指定某个任务在主线程最早可得的空闲时间执行，也就是说，尽可能早得执行。它在"任务队列"的尾部添加一个事件，因此要等到同步任务和"任务队列"现有的事件都处理完，才会得到执行。

HTML5标准规定了setTimeout()的第二个参数的最小值（最短间隔），不得低于4毫秒，如果低于这个值，就会自动增加。在此之前，老版本的浏览器都将最短间隔设为10毫秒。另外，对于那些DOM的变动（尤其是涉及页面重新渲染的部分），通常不会立即执行，而是每16毫秒执行一次。这时使用requestAnimationFrame()的效果要好于setTimeout()。
需要注意的是，setTimeout()只是将事件插入了"任务队列"，必须等到当前代码（执行栈）执行完，主线程才会去执行它指定的回调函数。要是当前代码耗时很长，有可能要等很久，所以并没有办法保证，回调函数一定会在setTimeout()指定的时间执行。

Node.js的Event Loop

node也是单线程的event loop,但是它的运行机制不同于浏览器环境。



1. V8引擎解析JavaScript脚本
2. 解析后的代码，调用node api
3. libuv库负责Node API的执行，它将不同的任务分配给不同的线程，形成一个Event Loop（事件循环），以异步的方式将任务的执行结果返回给V8引擎。
4. V8引擎再将结果返回给用户。

process.nextTick方法可以在当前执行栈的尾部，下一次event loop主线程读取任务队列之前触发回调函数，指定的任务会发生在所有异步任务之前。
setImmediate方法则是在当前任务队列的尾部添加事件，也就是它指定的任务总是在下一次Event Loop时执行，与setTimeout(fn,0)很像。