**关注”千锋互联”微信公众号( ID: qianfengjiaoyu ),更多学习资源(电子书/面试题/简历模板/视频/公开课等等)与你分享**

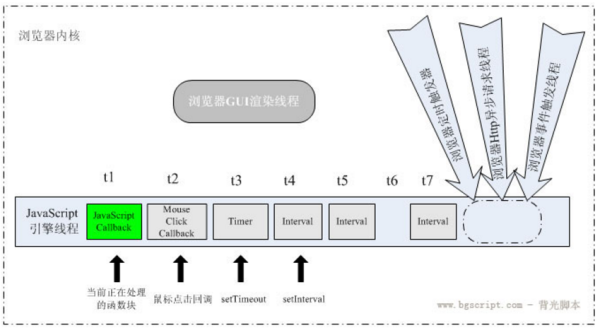


**Js中的异步与单线程(详解)**

说到JavaScript的单线程（single threaded）和异步（asynchronous），很多同学不禁会想，**这不是自相矛盾么**？其实，单线程和异步确实不能同时成为一个语言的特性。js选择了成为单线程的语言，所以它本身不可能是异步的，但js的宿主环境（比如浏览器，Node）是多线程的，宿主环境通过某种方式（事件驱动，下文会讲）使得js具备了异步的属性。往下看，你会发现js的机制是多么的简单高效！

**说说浏览器**

js是单线程语言，浏览器只分配给js一个主线程，用来执行任务（函数），但一次只能执行一个任务，这些任务形成一个任务队列排队等候执行，但前端的某些任务是非常耗时的，比如网络请求，定时器和事件监听，如果让他们和别的任务一样，都老老实实的排队等待执行的话，执行效率会非常的低，甚至导致页面的假死。所以，浏览器为这些耗时任务开辟了另外的线程，主要包括http请求线程，浏览器定时触发器，浏览器事件触发线程，这些任务是异步的。下图说明了浏览器的主要线程。

图片来自popAnt 画得太好，忍不住引过来 （[javascript 的线程问题](https://link.zhihu.com/?target=http%3A//blog.csdn.net/kfanning/article/details/5768776)）

**再说说任务队列**

刚才说到浏览器为网络请求这样的异步任务单独开了一个线程，那么问题来了，这些异步任务完成后，主线程怎么知道呢？答案就是回调函数，整个程序是事件驱动的，每个事件都会绑定相应的回调函数，举个栗子，有段代码设置了一个定时器

setTimeout(function(){

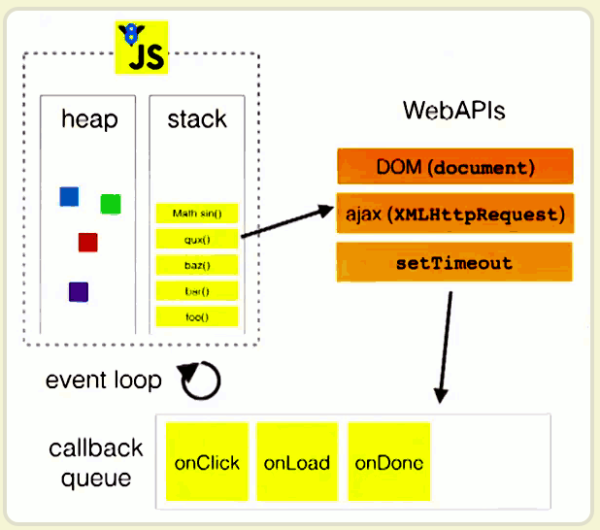
console.log(time is out);

}，50）;

执行这段代码的时候，浏览器异步执行计时操作，当50ms到了后，会触发定时事件，这个时候，就会把回调函数放到任务队列里。整个程序就是通过这样的一个个事件驱动起来的。  
所以说，js是一直是单线程的，浏览器才是实现异步的那个家伙。

**说回主线程**

js一直在做一个工作，就是从任务队列里提取任务，放到主线程里执行。下面我们来进行更深一步的理解。



图片来自Philip Roberts的演讲《Help, I'm stuck in an event-loop》非常深刻！

我们把刚才了解的概念和图中做一个对应，上文中说到的浏览器为异步任务单独开辟的线程可以统一理解为WebAPIs，上文中说到的任务队列就是callback queue，我们所说的主线程就是有虚线组成的那一部分，堆（heap）和栈（stack）共同组成了js主线程，函数的执行就是通过进栈和出栈实现的，比如图中有一个foo()函数，主线程把它推入栈中，在执行函数体时，发现还需要执行上面的那几个函数，所以又把这几个函数推入栈中，等到函数执行完，就让函数出栈。等到stack清空时，说明一个任务已经执行完了，这时就会从callback queue中寻找下一个人任务推入栈中（**这个寻找的过程，叫做event loop，因为它总是循环的查找任务队列里是否还有任务**）。

**借以解释几个容易困惑的问题**

**1、setTimeout(f1,0)是什么鬼**

这个语句最大的疑问是，f1是不是立刻执行？答案是不一定，因为要看主线程内的命令是否已经执行完了，如下代码：

setTimeout(function(){

console.log(1);

},0);

console.log(2);

**2、Ajax请求是否异步**

了解完上文内容，我们就知道了，ajax请求内容的时候是异步的，当请求完成后，会触发请求完成的事件，然后把回调函数放入callback queue，等到主线程执行该回调函数时还是单线程的。

**3、界面渲染线程是单独开辟的线程，是不是DOM一变化，界面就立刻重新渲染？**

如果DOM一变化，界面就立刻重新渲染，效率必然很低，所以浏览器的机制规定界面渲染线程和主线程是互斥的，主线程执行任务时，浏览器渲染线程处于挂起状态。

**如何利用浏览器的异步机制**

我们已经知道，js一直是单线程执行的，浏览器为几个明显的耗时任务单独开辟线程解决耗时问题，但是js除了这几个明显的耗时问题外，可能我们自己写的程序里面也会有耗时的函数，这种情况怎么处理呢？我们肯定不能自己开辟单独的线程，但我们可以利用浏览器给我们开放的这几个窗口，浏览器定时器线程和事件触发线程是好利用的，网络请求线程不适合我们使用。下面我们具体看一下：

假设耗时函数是f1，f1是f2的前置任务。

* 利用定时器触发线程

function f1(callback){

setTimeout(function(){

// f1 的代码

callback();

},0);

}

f1(f2);

**这种写法的耦合度高。**

* 利用事件触发线程

$f1.on('custom',f2); //这里绑定事件以jQuery写法为例

function f1(){

setTimeout(function(){

// f1的代码

$f1.trigger('custom');

},0);

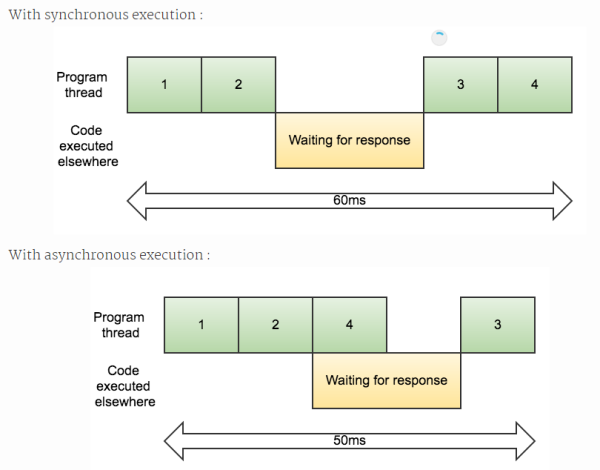
}

这种方法通过绑定自定义事件，对方法一解耦，这样可以通过绑定不同的事件，实现不同的回调函数，但如果应用这种方法过多，不利于阅读程序。

**异步的好处和适合的场景**

* **异步的好处**

我们直接通过一个例子对同步和异步进行对比，假设有四个任务（编号为1,2,3,4），它们的执行时间都是10ms,其中任务2是任务3的前置任务，任务2需要20ms的响应时间。下面我们做下对比，你就知道怎么实现的非阻塞I/O了。

图片来自Soham Kaman的[文章](https://link.zhihu.com/?target=http%3A//www.sohamkamani.com/blog/2016/03/14/wrapping-your-head-around-async-programming/" \t "_blank)

* **适合的场景**

可以看出，当我们的程序需要大量I/O操作和用户请求时，js这个具备单线程，异步，事件驱动多种气质的语言是多么应景！相比于多线程语言，它不必耗费过多的系统开销，同时也不必把精力用于处理多线程管理，相比于同步执行的语言，宿主环境的异步和事件驱动机制又让它实现了非阻塞I/O，所以你应该知道它适合什么样的场景了吧！

**关注”千锋互联”微信公众号( ID: qianfengjiaoyu ),更多学习资源(电子书/面试题/简历模板/视频/公开课等等)与你分享**

