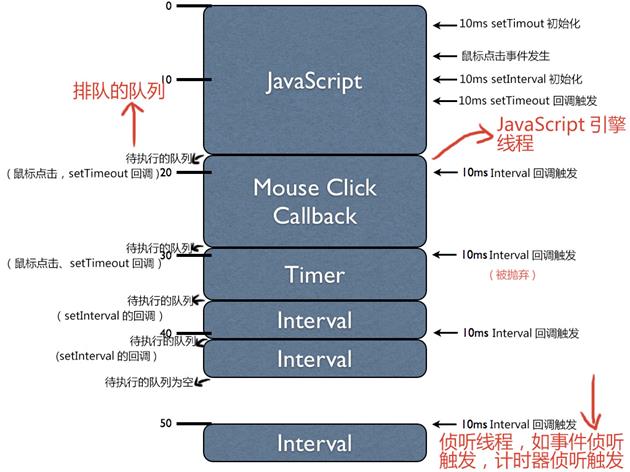
**JavaScript到底是不是单线程**

**http://www.cnblogs.com/wancy86/p/js\_threed.html**

**JavaScript引擎**

在了解计时器内部运作前，我们必须清楚一点，触发和执行并不是同一概念，计时器的回调函数一定会在指定delay的时间后被触发，但并不一定立即执行，可能需要等待。所有JavaScript代码是在一个线程里执行的，像鼠标点击和计时器之类的事件只有在JS单线程空闲时才执行。

我们来看一下图表，一开始你可能并没发现什么或啥都不懂，但请静下心来，在脑海里绘制出这个场景



这个图表中有许多数据信息等着我们去理解，当你完全理解了这个图，你会对js的异步运行机制（即JavaScript引擎如何实现异步事件）有很好的了解。这个图是一维的，垂直线上是以毫秒计位，蓝色块代表被划分的不同的js区域执行代码。例如，第一个JS区块执行了18毫秒，鼠标点击事件被阻塞了将近11毫秒，等等。

由于JavaScript引擎同一时间只执行一段代码（这是由JavaScript单线程的性质决定的），所以每个JS代码块阻塞了其它异步事件的进行。这意味着当一个异步事件（像鼠标点击、计时器、Ajax）发生时，这些事件的回调函数将排在队列后面等待执行（如何排队完全取决于各浏览器，而我们可以忽视它们内部差异，作一个简化处理）。

我们首先从第一个JS代码块开始，有两个计时器被初始化：一个10ms的setTimeout和一个10ms的setInterval.观察计时器初始化位置，（计时器初始化完毕后就会开始计时），发现setTimeout计时器的回调实际上会在第一个代码块执行完毕前被触发。但是这里注意的是，它不会立即执行（单线程不能这样做）。实际上，触发的回调将被排成一个队列，等待下一个可执行时间。

此外，在第一个JS代码块，我们发现一个鼠标点击事件被触发。这个鼠标点击JS回调被绑定在异步队列上（我们从来不知道用户什么时候执行这个操作，所以它被认为是异步的）且不能马上执行。像初始化的计时器一样,排队等待执行。

执行完初始化JS代码块后，浏览器就有个疑问：谁在等待执行？此时，鼠标点击回调和setTimeout计时器的回调都在等待。浏览器将选一个（鼠标点击事件）并立马执行。而计时器的回调将等待下一合适时机执行。

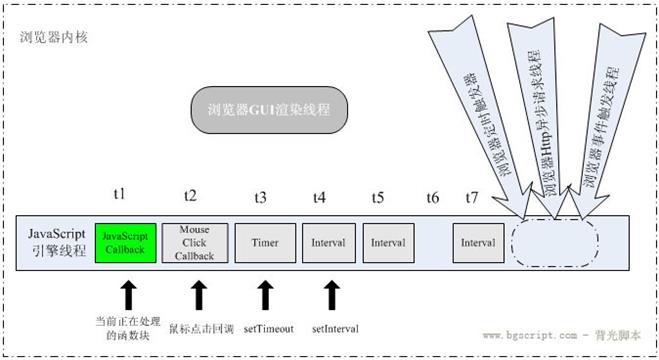
注意，鼠标点击事件执行过程中，interval的回调第一次被触发，与setTimeout的回调一样，排队等待执行。随着时间推移，等到setTimeout计时器的回调执行时候，setInterval的回调再次被触发，这次被触发的回调将被抛弃。如果一大段代码块正在执行，所有的setInterval的回调都将要排队，一旦大段代码块执行完毕，这些一连串的setInterval的回调相互间将被无延迟地执行。实际上，浏览器处理setInterval被触发的回调排队等待执行时，除非队列中setInterval回调为空，才允许新的setInterval的回调加入。

我们发现，setInterval的第一个被触发的回调执行时，setInterval的回调又被触发且排到队列。这向我们传达一个重要的消息：setInterval不关心目前JS正在执行的内容，setInterval的被触发的回调都将会无差别地排队。

最后，当setInterval的回调执行两次后，我们发现没有javascript引擎要执行东西。这意味着浏览器将等待着一个新的异步事件发生。我们知道，在50ms时候，setInterval的回调再次被触发，但这次并没有东西阻塞，所以回调就立马执行了。

在浏览器中，JavaScript引擎是基于事件驱动的，这里的事件可看作是浏览器派给它的各种任务，这些任务可能源自当前执行的代码块，如调用setTimeout()，也可能来自浏览器内核，如onload()、onclick()、onmouseover()、setTimeOut()、setInterval()、Ajax等。如果从代码的角度来看，所谓的任务实体就是各种回调函数，由于“单线程”的原因，这些任务会进行排队，一个接着一个等待着被引擎处理。(这段说法来源于http://www.benben.cc/blog/?p=327)

**JavaScript引擎线程和其它侦听线程**



上图中，定时器和事件都按时触发了，这表明JavaScript引擎的线程和计时器触发线程、事件触发线程是三个单独的线程，即使JavaScript引擎的线程被阻塞，其它两个触发线程都在运行。

浏览器内核实现允许多个线程异步执行，这些线程在内核制控下相互配合以保持同步。假如某一浏览器内核的实现至少有三个常驻线程： JavaScript引擎线程，事件触发线程，Http请求线程，下面通过一个图来阐明单线程的JavaScript引擎与另外那些线程是怎样互动通信的。虽然每个浏览器内核实现细节不同,但这其中的调用原理都是大同小异。

线程间通信：JavaScript引擎执行当前的代码块,其它诸如setTimeout给JS引擎添加一个任务,也可来自浏览器内核的其它线程,如界面元素鼠标点击事件,定时触发器时间到达通知,异步请求状态变更通知等.从代码角度看来任务实体就是各种回调函数,JavaScript引擎一直等待着任务队列中任务的到来.由于单线程关系,这些任务得进行排队,一个接着一个被引擎处理.

GUI渲染也是在引擎线程中执行的，脚本中执行对界面进行更新操作,如添加结点,删除结点或改变结点的外观等更新并不会立即体现出来,这些操作将保存在一个队列中,待JavaScript引擎空闲时才有机会渲染出来。来看例子（这块内容还有待验证,个人觉得当Dom渲染时，才可阻止渲染）

<div id="test">test</div>

<script type="text/javascript" language="javascript">

var i=0;

while(1) {

document.getElementById("test").innerHTML+=i++ + "<br />";

}

</script>

这段代码的本意是从0开始顺序显示数字，它们将一个接一个出现，现在我们来仔细研究一下代码，while(1)创建了一个无休止的循环，但是对于单线程的JavaScript引擎而言，在实际情况中就会造成浏览器暂停响应并处于假死状态。

alert()会停止JS引擎的执行，直到按确认键，在JS调试的时候，查看当前实时页面的内容。

**Ajax时如何实现异步的**

很多同学朋友搞不清楚，既然说JavaScript是单线程运行的，那么XMLHttpRequest在连接后是否真的异步？其实请求确实是异步的，不过这请求是由浏览器新开一个线程请求(参见上图)，当请求的状态变更时，如果先前已设置回调，这异步线程就产生状态变更事件放到JavaScript引擎的处理队列中等待处理，当任务被处理时，JavaScript引擎始终是单线程运行回调函数，具体点即还是单线程运行onreadystatechange所设置的函数。

原文链接： <http://www.cnblogs.com/sprying/archive/2013/05/26/3100639.html>

分类: [javascript](http://www.cnblogs.com/wancy86/category/526214.html)