http://www.2cto.com/kf/201204/129337.html

1.先review一下我在live800上面的坑

麻烦请看这两行代码：

1）.G_LIVE_OBJ.sent = setTimeout(index.queryLiveMsg, G_LIVE_OBJ.sentTime);

2）.G_LIVE_OBJ.sent = setTimeout(index.queryLiveMsg(), G_LIVE_OBJ.sentTime);

有人知道他们的区别么？

demo代码：

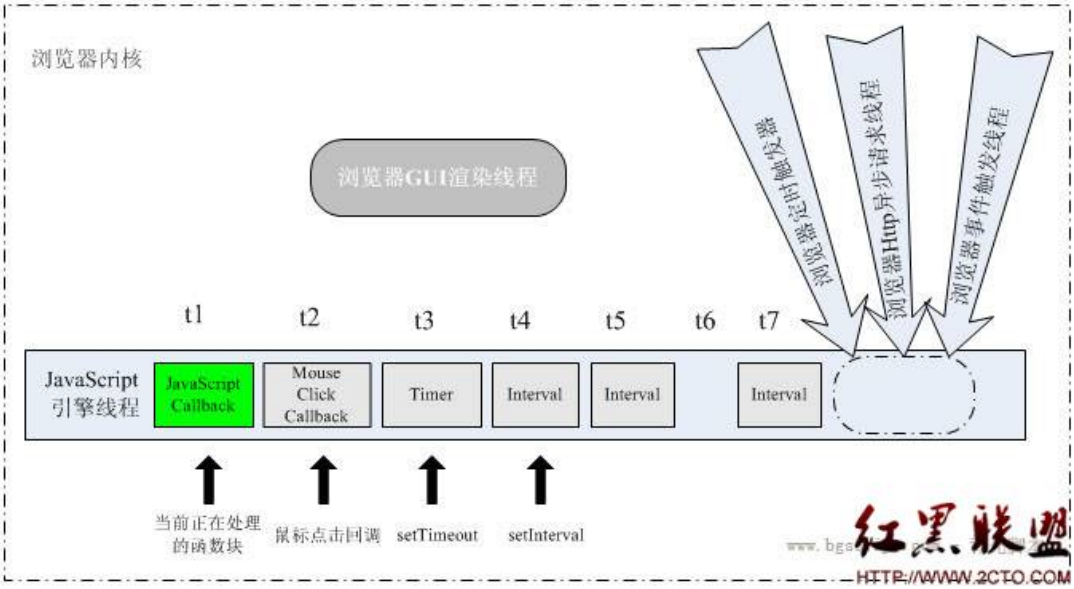
```
var a = function(){alert('a')},
    b = function(){alert('b')};

setTimeout(a(),10000);
setTimeout(b,1000);
```

setTimeout里面必须是需要**编译的代码，或者一个函数方法**，不能传入一段**可执行的代码**，否则会立即执行

2.上面这个结论是我百度了很久之后才得出的结论，在得出这个结论的过程中，由于不是很懂setTimeout的原理，就顺手百度了一下js的执行序列，即setTimeout的工作原理，感觉执行序列这个对我们平时工作挺有帮助的，及时分享一下：

1）js的执行是单线程的，所有事件执行，都是建立在这个单线程的基础上面。单线程是什么意思呢，就是js里面所有的事件都是需要排队，入栈执行的。然后，js本质上是有两个串行的队列，主队列和任务队列。当且仅当主队列完成之后，任务队列里面的时间才会依次执行。setTimeout，setInterval，异步回调事件，鼠标点击回调事件等，这些事件在主队列执行的时候，会被依次加入任务队列。等主队列中的事件执行完毕，任务队列的事件才开始执行。

2）可以先看一下一下这个图



这里可以看出：正在处理的函数块是在，鼠标点击回调，setTimeout,setInterval,异步请求回调这几种事件之前的。

任务队列里面的事件是等js线程有空的时候再执行的，若线程卡死，则不会进行下去。代码如下：

```
var a = function(){ while(true){}},
    b = function(){alert(123);};

    setTimeout(a, 0);
    b();
```

此时执行会alert123，然后再浏览器直接卡死。这就很清楚的告诉了我们，线程先执行了b()再执行setTimeout里面的内
容。

一些小的特性：

渲染进程：
js线程是会阻塞页面渲染线程的，页面的渲染都必须等js代码执行完毕才会继续渲染，浏览器会识别js线程前后的页面元素的区别，再去渲染页面，假如你在js中，把一个元素修改之后，再恢复原来的样子，浏览器是会默认不对这个元素做任何操作，以为对到浏览器来说，这个元素并未发生改变。因此渲染进程会被挂起的。（自己写了个例子，发现跟说好的不一样，看看各位大神的意见）

事件触发进程：
由上图可以看出来所有在js线程中被触发的事件，都会被放在队列的最后面，在主队列完成后，任务队列的事件，才会一个一个的执行
```
$('#s_tab').on('click', function(){alert(1)});
var a = function(){var i = 0; while(i<20000000000){i++} };a();
```

所以一次鼠标点击，或是计时器到达时间点，或是Ajax请求完成触发了回调函数，这些事件处理程序或回调函数都不会立即运行，而是立即排队，一旦线程有空闲就执行。