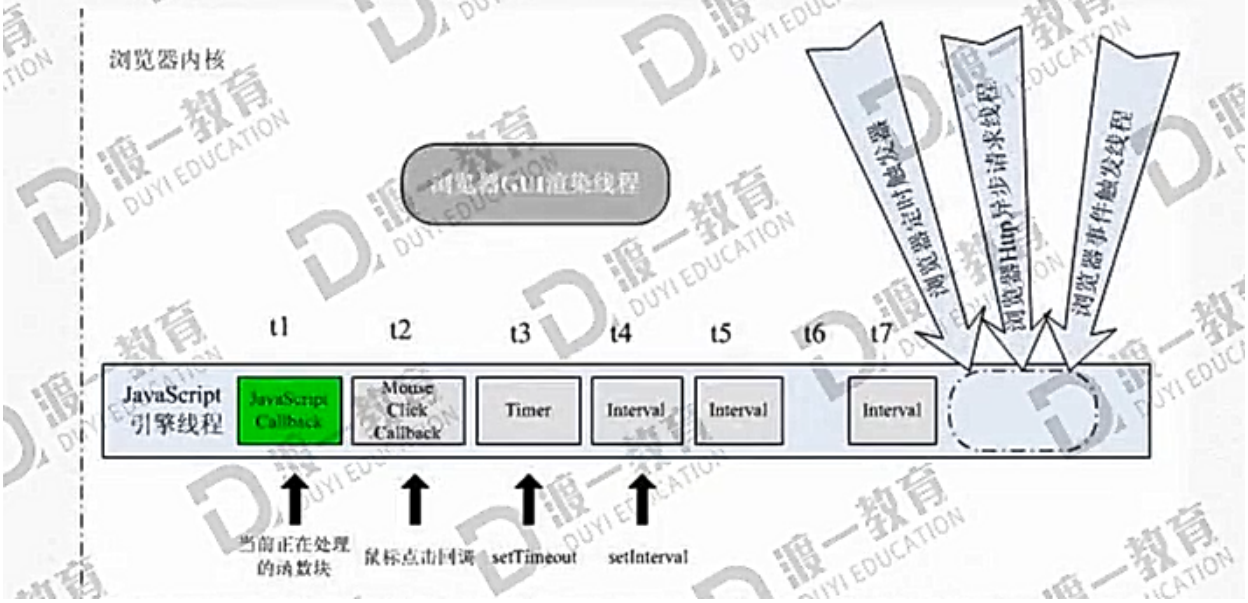
浏览器常驻的线程

- js引擎线程 （解释执行js代码、用户输入、网络请求）
- GUI线程 （绘制用户界面、与js主线程是互斥的）
- http网络请求线程 （处理用户的get、post等请求，等返回结果后将回调函数推入任务队列）
- 定时触发器线程 （setTimeout、setInterval等待时间结束后把执行函数推入任务队列中）
- 浏览器事件处理线程 （将click、mouse等交互事件发生后将这些事件放入事件队列中）

UI主线程负责协调运转



JS引擎线程和GUI线程-互斥

- JS可以操作DOM元素，进而会影响到GUI的渲染结果，因此**JS引擎线程与GUI渲染线程是互斥的**。也就是说当JS引擎线程处于运行状态时，GUI渲染线程将处于冻结状态。

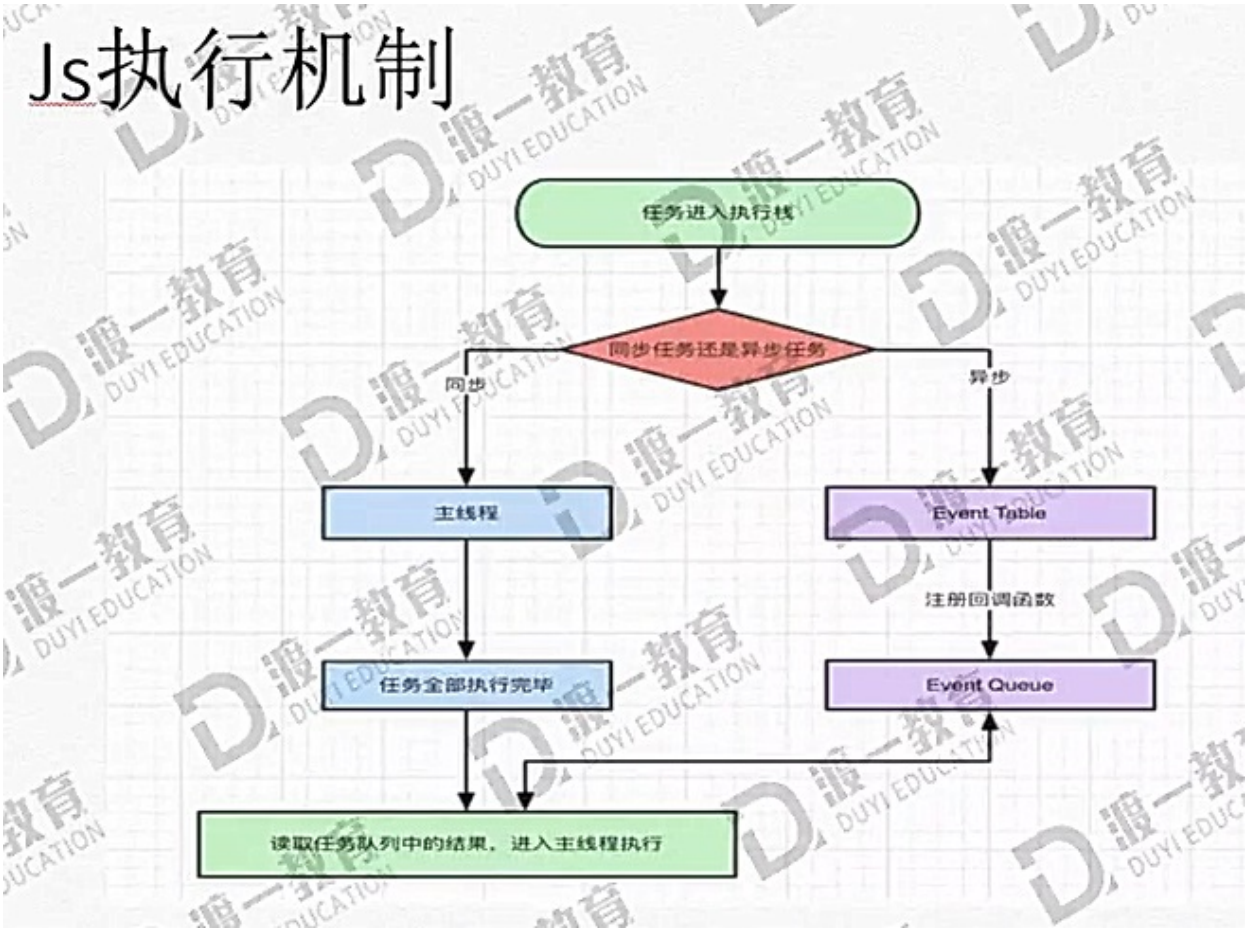
也就是说执行时，点击的话是不会有图像变化的

Js执行机制-多线程不好吗？

- js设计出来就是为了与用户交互，处理DOM，假如js是多线程，同一时间一个线程想要修改DOM，另一个线程想要删除DOM，问题就变得复杂许多，浏览器不知道听谁的，如果引入“锁”的机制，这不就又回到了被其他语言尴尬的困境了吗。

Js执行机制

JavaScript是基于单线程运行的，同时又是可以异步执行的，一般来说这种既是单线程又是异步的语言都是基于事件来驱动的，恰好浏览器就给**JavaScript**提供了这么一个环境



因此要执行定时器（时间到了只是把任务放入事件队列）等异步任务时，要等主线程内的任务全部完成，所以有时候定时器会不准

重新理解定时器

setTimeout的等待时间结束后并不是直接执行的而是先推入浏览器的一个任务队列，在同步队列结束后在依次调用任务队列中的任务。

setTimeout(function(){}, 0)Js主线程中的执行栈为空时，0毫秒实际上也达不到的，根据HTML标准，最低4毫秒。

setInterval是每隔一段时间把任务放到Event Queue之中

同步任务

0.代码没有执行的时候，执行栈为空栈

1.foo函数执行时，创建了一帧，这帧中包含了形参、局部变量（预编译过程），然后把这一帧压入栈中

2.然后执行foo函数内代码，执行bar函数

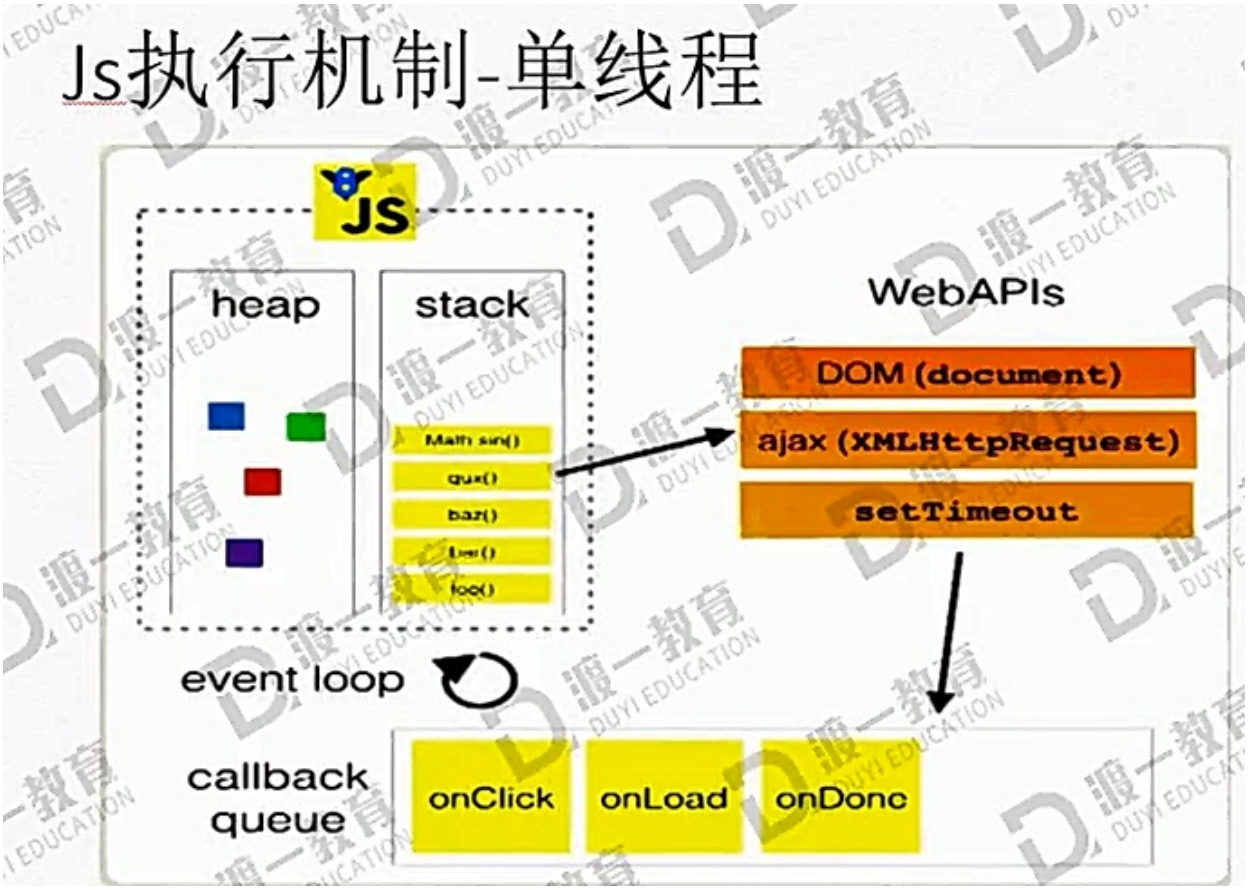
3.创建新帧，同样有形参、局部变量，压入栈中

4.bar函数执行完毕，弹出栈

5.foo函数执行完毕，弹出栈

6.执行栈为空

执行栈其实相当于js主线程



其中JS虚线框就是主线程，stack是执行栈