setTimeout 与 requestAnimationFrame 的区别

提到 setTimeout 以及 requestAnimationFrame，大家的第一反应是动画相关的两个 API。

我们来谈谈什么是动画。动画其实是一种假象，是一种不连续的运动以帧的形式呈现给我们的东西。在二十世纪，通常人们观看的电影其实就是通过胶片记录和投影的。它是以每秒至少 24 帧的速度形成的视觉上的运动起来的假象。NTSC 广播的标准的帧速率为 23.975FPS，而 PAL 制式的为 25FPS。

FPS 可以理解为我们常说的“刷新率（单位为 Hz）”，例如我们常在游戏里说的“FPS 值”。我们在装机选购显卡和显示器的时候，都会注意到“刷新率”。一般我们设置缺省刷新率都在 75Hz（即 75 帧/秒）以上。例如：75Hz 的刷新率刷也就是指屏幕一秒内只扫描 75 次，即 75 帧/秒。而当刷新率太低时我们肉眼都能感觉到屏幕的闪烁，不连贯，对图像显示效果和视觉感观产生不好的影响。

因此，至少要以 24FPS 的速率才能形成动画，但这样的动画并不是平滑的，流畅的。平滑的动画要以无线帧速率才能实现，但是对于人类大脑而言是侦测不到那种情况下的帧速率，可以说 60FPS 就已经很不错了。常见的电脑、智能手机等大部分现代化设备通常是以 60FPS 的速率刷新屏幕的，少部分游戏系统则支持 120FPS。

那么，什么又是帧呢？这个没有绝对的定义，它主要是依赖于使用的具体环境。例如，电影胶片的每一帧都是由所记录的 FPS 决定的。在录制视频时，把摄像机的帧率调为 30FPS，那么就必须以 30FPS 的速率在 1s 内播放生成的 30 个单独图像。然而，在讨论 web 时，帧的定义又发生了变化。

对于 web 动画，我们可以在设备屏幕中移动 1px 或者更多。移动一个元素（DOM 元素）的像素越少，那么动画就越流畅，越平滑。帧其实就是 DOM 元素在屏幕上的实时位置的一个快照。在 1s 内，如果一个元素以 1px/次 的速度移动 60px，那么 FPS 值就是 60。也就是说，上面等价于以 2px/次 的速度移动 120px。虽然移动速度变大了，但是动画并不会更加流畅平滑，因为相应的元素的移动距离也变大了。

那么，如何使用 JavaScript 让 DOM 元素产生动画效果呢？

可以使用 JavaScript 中的 setTimeout 或者 setInterval 函数。

setTimeout 是以 n 毫秒后执行回调函数，回调函数中可以递归调用 setTimeout 来实现动画。

setInterval 以 n 毫秒的间隔时间调用回调函数。

为了实现 60FPS，我们需要以 60 次/s 的速度移动一个元素，那意味着元素必须移动大约 16.7ms（1000ms/60frames）。

上面的数据是我们理论计算出来的数据。当我们执行

```
setTimeout(function () {}, 1000/60)
```

这段代码的的时候，它真的是在 1000/60 毫秒后执行的么？了解 setTimeout 机制的同学会知道，**不一定会在 1000/60 毫秒后执行，可能会更长**。那这是为什么呢？

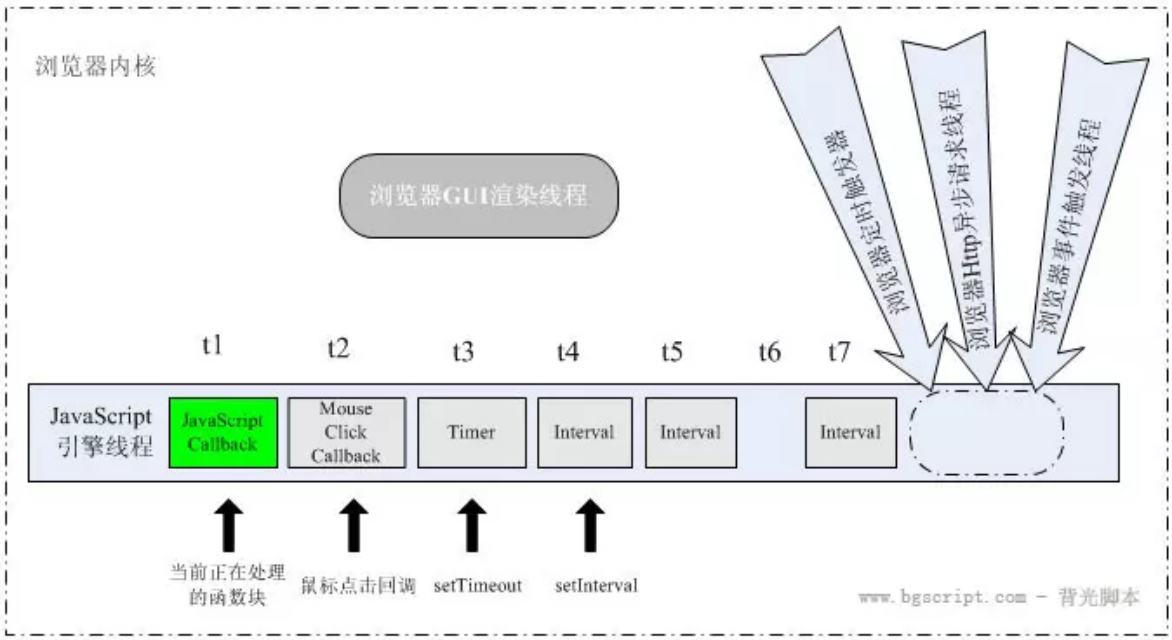
JavaScript 中的事件队列

众所周知，Javascript 引擎（以下简称 JS 引擎）是单线程的，在某一个特定的时间内只能执行一个任务，并阻塞其他任务的执行，也就是说这些任务是串行的。

这样的话，用户不得不等待一个耗时的操作完成之后才能进行后面的操作，这显然是不能容忍的，但是实际开发中我们却可以使用异步代码来解决。

当异步方法比如这里的 setTimeout()，或者 ajax 请求、DOM 事件执行的时候，会交由浏览器内核的其他模块去管理。

当异步的方法满足触发条件后，该模块就会将方法推入到一个任务队列中，当主线程代码执行完毕处于空闲状态的时候，就会去检查任务队列，将队列中第一个任务入栈执行，完毕后继续检查任务队列，如此循环。前提条件是主线程处于空闲状态，这就是事件循环的模型。



js 引擎单线程执行的,它是基于事件驱动的语言.它的执行顺序是遵循一个叫做事件队列的机制.从图中我们可以看出,浏览器有各种各样的线程,比如事件触发器,网络请求,定时器等等.线程的联系都是基于事件的.js 引擎处理到与其他线程相关的代码,就会分发给其他线程,他们处理完之后,需要 js 引擎计算时就是在事件队列里面添加一个任务. 这个过程中,js 并不会阻塞代码等待其他线程执行完毕,而且其他线程执行完毕后添加事件任务告诉 js 引擎执行相关操作.这就是 js 的异步编程模型.

在指定时间内, 将任务放入事件队列,等待 js 引擎空闲后被执行.

这样就好解释了，为什么说好的 16 秒后执行，但是可能在 18 秒执行，是因为 js 引擎正在处理其他内容。

setTimeout/setInterval 不单是不准时的问题。还有其他问题，总结如下：

- 动画作者对帧数没有掌控；
- 当标签是隐藏状态（非当前显示的）时，无谓地消耗系统资源；
- setInterval 对自己调用的代码是否报错漠不关心。即使调用的代码报错了，它依然会持续的调用下去（可以用 setTimeout 解决）

由于上面种种问题。Mozilla 的哥们说：“不如咱好心弄个 API，让那些做动画的民工门不再用那该死的 setInterval 了。大家（Chrome，IE，还有那个挪威的哥们）一致同意。

于是，我们有了 requestAnimationFrame

requestAnimationFrame

HTML5 新增加的 API，类似于 setTimeout 定时器。window 对象的一个方法，window.requestAnimationFrame

浏览器(所以只能在浏览器中使用)专门为动画提供的 API，让 DOM 动画、Canvas 动画、SVG 动画、WebGL 动画等有一个统一的刷新机制。

特点：

- **按帧对网页进行重绘**。该方法告诉浏览器希望执行动画并请求浏览器在下一次重绘之前调用回调函数来更新动画
- **由系统来决定回调函数的执行时机**，在运行时浏览器会自动优化方法的调用

- 显示器有固定的刷新频率（60Hz 或 75Hz），也就是说，每秒最多只能重绘 60 次或 75 次，requestAnimationFrame 的基本思想**让页面重绘的频率与这个刷新频率保持同步**

  比如显示器屏幕刷新率为 60Hz，使用 requestAnimationFrame API，那么回调函数就每 1000ms / 60 ≈ 16.7ms 执行一次；如果显示器屏幕的刷新率为 75Hz，那么回调函数就每 1000ms / 75 ≈ 13.3ms 执行一次。

- 通过 requestAnimationFrame 调用回调函数引起的页面重绘或回流的时间间隔和显示器的刷新时间间隔相同。所以 requestAnimationFrame 不需要像 setTimeout 那样传递时间间隔，而是浏览器通过系统获取并使用显示器刷新频率

优势：

从实现的功能和使用方法上，requestAnimationFrame 与定时器 setTimeout 都相似，所以说其优势是同 setTimeout 实现的动画相比。

a. 提升性能，防止掉帧

- 浏览器 UI 线程：浏览器让执行 JavaScript 和更新用户界面（包括重绘和回流）共用同一个单线程，称为“浏览器 UI 线程”
- 浏览器 UI 线程的工作基于一个简单的队列系统，任务会被保存到队列中直到进程空闲。一旦空闲，队列中的下一个任务就被重新提取出来并运行。这些任务要么是运行 JavaScript 代码，要么执行 UI 更新。

setTimeout 通过设置一个间隔时间不断改变图像，达到动画效果。该方法在一些低端机上会出现卡顿、抖动现象。这种现象一般有两个原因：

1. setTimeout 的执行时间并不是确定的。
2. 刷新频率受屏幕分辨率和屏幕尺寸影响，不同设备的屏幕刷新率可能不同，setTimeout 只能设置固定的时间间隔，这个时间和屏幕刷新间隔可能不同
3. 以上两种情况都会导致 setTimeout 的执行步调和屏幕的刷新步调不一致，从而引起丢帧现象。

使用 requestAnimationFrame 执行动画，最大优势是能保证回调函数在屏幕每一次刷新间隔中只被执行一次，这样就不会引起丢帧，动画也就不会卡顿。

b. 节约资源，节省电源

- 使用 setTimeout 实现的动画，当页面被隐藏或最小化时，定时器 setTimeout 仍在后台执行动画任务，此时刷新动画是完全没有意义的（实际上 FireFox/Chrome 浏览器对定时器做了优化：页面闲置时，如果时间间隔小于 1000ms，则停止定时器，与 requestAnimationFrame 行为类似。如果时间间隔>=1000ms，定时器依然在后台执行）
- 使用 requestAnimationFrame，当页面处于未激活的状态下，该页面的屏幕刷新任务会被系统暂停，由于 requestAnimationFrame 保持和屏幕刷新同步执行，所以也会被暂停。当页面被激活时，动画从上次停留的地方继续执行，节约 CPU 开销。

了解一下 GUI 引擎（渲染引擎）

之前提到了 js 引擎,就不得不说浏览器的另外一个引擎---GUI 渲染引擎. 在 js 中渲染操作也是异步的.比如 dom 操作的代码会在事件队列中生成一个任务,js 执行到这个任务时就会去调用 GUI 引擎渲染。

js 语言设定 js 引擎与 GUI 引擎是互斥的,也就是说 GUI 引擎在渲染时会阻塞 js 引擎计算.原因很简单,如果在 GUI 渲染的时候,js 改变了 dom,那么就会造成渲染不同步。

总结

setTimeout 与 requestAnimationFrame 的区别：

- **引擎层面**：setTimeout 属于 JS 引擎，存在事件轮询，存在事件队列。
  requestAnimationFrame 属于 GUI 引擎，发生在渲染过程的中重绘重排部分，与电脑分辨路保持一致。
- **性能层面**：当页面被隐藏或最小化时，定时器 setTimeout 仍在后台执行动画任务。
  当页面处于未激活的状态下，该页面的屏幕刷新任务会被系统暂停，requestAnimationFrame 也会停止。
- **应用层面**：利用 setTimeout，这种定时机制去做动画，模拟固定时间刷新页面。
  requestAnimationFrame 由浏览器专门为动画提供的 API，在运行时浏览器会自动优化方法的调用，在特定性环境下可以有效节省了 CPU 开销。