getContext（）

getContext() 方法返回一个用于在画布上绘图的环境。

Random（）

random() 方法可返回介于 0 ~ 1 之间的一个随机数。

beginPath()

beginPath() 方法开始一条路径，或重置当前的路径。

strokeStyle

strokeStyle 属性设置或返回用于笔触的颜色、渐变或模式。

Stroke()

stroke() 方法会实际地绘制出通过 moveTo() 和 lineTo() 方法定义的路径。默认颜色是黑色。

requestAnimationFrame

在Web应用中，实现动画效果的方法比较多，Javascript 中可以通过定时器 setTimeout 来实现，css3 可以使用 transition 和 animation 来实现，html5 中的 canvas 也可以实现。除此之外，html5 还提供一个专门用于请求动画的API，那就是 requestAnimationFrame，顾名思义就是****请求动画帧。****为了深入理解 requestAnimationFrame 背后的原理，我们首先需要了解一下与之相关的几个概念：

#### **1、屏幕刷新频率**

即图像在屏幕上更新的速度，也即屏幕上的图像每秒钟出现的次数，它的单位是赫兹(Hz)。 对于一般笔记本电脑，这个频率大概是60Hz， 可以在桌面上****右键->屏幕分辨率->高级设置->监视器**** 中查看和设置。这个值的设定受屏幕分辨率、屏幕尺寸和显卡的影响，原则上设置成让眼睛看着舒适的值都行。

市面上常见的显示器有两种，即****CRT****和****LCD****， CRT就是传统显示器，LCD就是我们常说的液晶显示器。

CRT是一种使用阴极射线管的显示器，屏幕上的图形图像是由一个个因电子束击打而发光的荧光点组成，由于显像管内荧光粉受到电子束击打后发光的时间很短，所以电子束必须不断击打荧光粉使其持续发光。****电子束每秒击打荧光粉的次数就是屏幕刷新频率。****

而对于LCD来说，则不存在刷新频率的问题，它根本就不需要刷新。因为LCD中每个像素都在持续不断地发光，直到不发光的电压改变并被送到控制器中，所以LCD不会有电子束击打荧光粉而引起的闪烁现象。

因此，****当你对着电脑屏幕什么也不做的情况下，显示器也会以每秒60次的频率正在不断的更新屏幕上的图像****。为什么你感觉不到这个变化？ 那是因为人的眼睛有****视觉停留效应****，即前一副画面留在大脑的印象还没消失，紧接着后一副画面就跟上来了，这中间只间隔了****16.7ms****(1000/60≈16.7)， 所以会让你误以为屏幕上的图像是静止不动的。而屏幕给你的这种感觉是对的，试想一下，如果刷新频率变成1次/秒，屏幕上的图像就会出现严重的闪烁，这样就很容易引起眼睛疲劳、酸痛和头晕目眩等症状。

#### **2、动画原理**

根据上面的原理我们知道，你眼前所看到图像正在以每秒60次的频率刷新，由于刷新频率很高，因此你感觉不到它在刷新。而****动画本质就是要让人眼看到图像被刷新而引起变化的视觉效果，这个变化要以连贯的、平滑的方式进行过渡。**** 那怎么样才能做到这种效果呢？

刷新频率为60Hz的屏幕每16.7ms刷新一次，我们在屏幕每次刷新前，将图像的位置向左移动一个像素，即1px。这样一来，屏幕每次刷出来的图像位置都比前一个要差1px，因此你会看到图像在移动；由于我们人眼的视觉停留效应，当前位置的图像停留在大脑的印象还没消失，紧接着图像又被移到了下一个位置，因此你才会看到图像在流畅的移动，这就是视觉效果上形成的动画。

#### ****3、setTimeout****

理解了上面的概念以后，我们不难发现，setTimeout 其实就是通过设置一个间隔时间来不断的改变图像的位置，从而达到动画效果的。但我们会发现，利用seTimeout实现的动画在某些低端机上会出现卡顿、抖动的现象。 这种现象的产生有两个原因：

* setTimeout的执行时间并不是确定的。在Javascript中， setTimeout 任务被放进了异步队列中，只有当主线程上的任务执行完以后，才会去检查该队列里的任务是否需要开始执行，因此 ****setTimeout 的实际执行时间一般要比其设定的时间晚一些。****
* 刷新频率受****屏幕分辨率****和****屏幕尺寸****的影响，因此不同设备的屏幕刷新频率可能会不同，而 setTimeout只能设置一个固定的时间间隔，这个时间不一定和屏幕的刷新时间相同。

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。 那为什么步调不一致就会引起丢帧呢？

首先要明白，setTimeout的执行只是在内存中对图像属性进行改变，这个变化必须要等到屏幕下次刷新时才会被更新到屏幕上。如果两者的步调不一致，就可能会导致中间某一帧的操作被跨越过去，而直接更新下一帧的图像。假设屏幕每隔16.7ms刷新一次，而setTimeout每隔10ms设置图像向左移动1px， 就会出现如下绘制过程：

* 第0ms: 屏幕未刷新，等待中，setTimeout也未执行，等待中；
* 第10ms: 屏幕未刷新，等待中，setTimeout开始执行并设置图像属性left=1px；
* 第16.7ms: 屏幕开始刷新，屏幕上的图像向左移动了****1px****， setTimeout 未执行，继续等待中；
* 第20ms: 屏幕未刷新，等待中，setTimeout开始执行并设置left=2px;
* 第30ms: 屏幕未刷新，等待中，setTimeout开始执行并设置left=3px;
* 第33.4ms:屏幕开始刷新，屏幕上的图像向左移动了****3px****， setTimeout未执行，继续等待中；
* …

从上面的绘制过程中可以看出，屏幕没有更新left=2px的那一帧画面，图像直接从1px的位置跳到了3px的的位置，这就是丢帧现象，这种现象就会引起动画卡顿。

#### **4、requestAnimationFrame**

与setTimeout相比，requestAnimationFrame最大的优势是****由系统来决定回调函数的执行时机。****具体一点讲，如果屏幕刷新率是60Hz,那么回调函数就每16.7ms被执行一次，如果刷新率是75Hz，那么这个时间间隔就变成了1000/75=13.3ms，换句话说就是，requestAnimationFrame的步伐跟着系统的刷新步伐走。****它能保证回调函数在屏幕每一次的刷新间隔中只被执行一次****，这样就不会引起丢帧现象，也不会导致动画出现卡顿的问题。

这个API的调用很简单，如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | **var** progress = 0;  *//回调函数*  **function** render() {      progress += 1; *//修改图像的位置*  **if** (progress < 100) {  *//在动画没有结束前，递归渲染*             window.requestAnimationFrame(render);      }  }  *//第一帧渲染*  window.requestAnimationFrame(render); |

除此之外，requestAnimationFrame还有以下两个优势：

* ****CPU节能****：使用setTimeout实现的动画，当页面被隐藏或最小化时，setTimeout 仍然在后台执行动画任务，由于此时页面处于不可见或不可用状态，刷新动画是没有意义的，完全是浪费CPU资源。而requestAnimationFrame则完全不同，当页面处理未激活的状态下，该页面的屏幕刷新任务也会被系统暂停，因此跟着系统步伐走的requestAnimationFrame也会停止渲染，当页面被激活时，动画就从上次停留的地方继续执行，有效节省了CPU开销。
* ****函数节流****：在高频率事件(resize,scroll等)中，为了防止在一个刷新间隔内发生多次函数执行，使用requestAnimationFrame可保证每个刷新间隔内，函数只被执行一次，这样既能保证流畅性，也能更好的节省函数执行的开销。一个刷新间隔内函数执行多次时没有意义的，因为显示器每16.7ms刷新一次，多次绘制并不会在屏幕上体现出来。

#### **5、优雅降级**

由于requestAnimationFrame目前还存在兼容性问题，而且不同的浏览器还需要带不同的前缀。因此需要通过优雅降级的方式对requestAnimationFrame进行封装，优先使用高级特性，然后再根据不同浏览器的情况进行回退，直止只能使用setTimeout的情况。下面的代码就是有人在github上提供的polyfill，详细介绍请参考github代码 [requestAnimationFrame](https://github.com/darius/requestAnimationFrame" \t "http://web.jobbole.com/91578/_blank)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | **if** (!Date.now)      Date.now = **function**() { **return** **new** Date().getTime(); };  (**function**() {      'use strict';    **var** vendors = ['webkit', 'moz'];  **for** (**var** i = 0; i < vendors.length && !window.requestAnimationFrame; ++i) {  **var** vp = vendors[i];          window.requestAnimationFrame = window[vp+'RequestAnimationFrame'];          window.cancelAnimationFrame = (window[vp+'CancelAnimationFrame']                                     || window[vp+'CancelRequestAnimationFrame']);      }  **if** (/iP(ad|hone|od).\*OS 6/.test(window.navigator.userAgent) *// iOS6 is buggy*          || !window.requestAnimationFrame || !window.cancelAnimationFrame) {  **var** lastTime = 0;          window.requestAnimationFrame = **function**(callback) {  **var** now = Date.now();  **var** nextTime = Math.max(lastTime + 16, now);  **return** setTimeout(**function**() { callback(lastTime = nextTime); },                                nextTime - now);          };          window.cancelAnimationFrame = clearTimeout;      }  }()); |