一文搞懂动画原理和requestAnimationFram...

前言

作为一名前端开发者，相信你一定接触过动画。还记得最开始学习前端时，我们曾尝试使用 JS 实现各种动画效果，比如轮播图等等。随着前端技术的不断更新，我们实现动画的方式变得多种多样了，比如使用 JS、亦或者使用 CSS 动画、又或者使用 canvas 等动画技术。

今天我们将要讲的便是 JS 中实现动画，了解 JS 中实现动画的方式和原理，以及各种优缺点。

最终我们将得到答案：**为什么我们的动画有时候会卡顿？如何解决卡顿现象？**

1.动画基础

在进入本篇文章的正题之前，我们有比较学习一下动画的基础知识，一遍我们后续能够找到问题的根源。

1.1 屏幕刷新率

相信大家现在买手机或者电脑都听到很多厂家在那儿吹嘘屏幕刷新率吧！比如 60Hz（赫兹）、90Hz、120Hz 等等。那么这个屏幕刷新率到底是什么意思呢?

我们可以给一个简单的定义，大家好理解一点：

> 屏幕刷新率即图像在屏幕上更新的速率，也可以理解为屏幕上图像每秒钟出现的次数。

比如我们的电脑屏幕刷新率为 60Hz，那么意味着我们的屏幕会以每秒 60 次的频率不断更新屏幕上的图像，意味着每大约 16.7ms（1000/60≈16.7）屏幕就会更新一张图片，虽然屏幕在不停的更新图片，但是这种更新我们普通人的眼睛是感觉不出来的，所以我们为觉得图像是连贯的。

大家想一想，如果我们把屏幕刷新率换为 1Hz，那么是不是每 1 秒更新一下屏幕画面，是不是就会觉得卡顿~~

查看屏幕刷新率：

大家可以通过电脑的显示**设置–>屏幕–>高级**设置查看自己电脑的刷新率。

1.2 动画的原理

我们知道了屏幕刷新率，那么我们就很容易知道动画实现的原理了。

需求：

假设在 60Hz 的屏幕下，我们想要实现一个 div 从左->右移动 100px，并且 div 平滑移动，而不是一下到达终点。

实现思路：

60Hz 的屏幕，意味着每 16.7ms 屏幕就会更新图像，如果我们在屏幕每次刷新图像前，将 div 元素向右移动 1px，这样的话，每次屏幕刷新的时候，我们的 div 元素都向右移动了一点点位置。但是由于间隔时间太短，我们的大脑和眼睛是无法处理的，所以我们会误认为 div 是平滑移动的。

所以我们的动画原理其实很简单，就是在很短的时间内不断更新相同间隔或者相同差别的图片，给我们人眼造成连续的假象。

我们常说的 60 帧电影，其实就是每秒 60 刷新 60 次。

2.setInterval 实现动画

我们知道了屏幕刷新率和动画的实现原理后，我们再来实现 div 从左向右移动这个动画就很简单了。

在 CSS3 动画出来以前，我们通常使用 JS 来实现动画效果，即使有了 CSS 动画，我们很多动画效果还是得依赖 JS。而 JS 实现动画得两大利器便是 setTimeout 和 setInterval，因为我们动画的原理就是不听的刷新图像，而定时器可以帮我们做这一操作。

2.1 setInterval 平滑动画

```
比如我们使用 setInterval 实现 div 从左往右
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

<title>一文搞懂 requestAnimationFrame 动画函数！</title>移动这一动画。

代码如下：

```
<head>
  <style>
    .box {
      width: 100px;
      height: 100px;
      background-color: blue;
      position: relative;

}
  </style>
</head>
<body>
```

```
  <div class="box"></div>
</body>
<script>
  let box = document.getElementsByClassName('box')[0];
  // 动画函数
  let num = 0;
  function animation() {
    if (num < 1000) {
      num++;
      box.style.left = num + 'px'
    } else {
      clearInterval(tim  }
  }
  // 开启循环定时器
  let timer = setInterval(animatio7);
</script>
```

实现效果：

上段代码中我们编写了一个动画函数，代码很简单，就是让 div 元素向右移动 1px。然后我们开启定时器不断重复该函数，需要注意的是，我们将函数执行的频率调为了差不多和屏幕刷新率一样，即 60Hz 的屏幕刷新率。

结合我们动画实现原理来看，相当于我们每次在屏幕刷新图像的时候 div 向右移动 1px，由于人眼看不出区别，所以我们觉得 div 移动是平滑的。

2.2 setInterval 非平滑动画

前面是我们是依照屏幕刷新率来设定 setInterval 执行间隔时间的，理想情况下动画是平滑的，但是我们如果把间隔时间稍微改一改，会发生什么效果呢?

修改代码如下：

```
let timer = setInterval(animation, 500);
```

实现效果：

从上图可以看出，我们将函数执行实践间隔调为了 500ms，意味着每 500msdiv 元素向前移动 1px，我们发现动画出现了明显的卡顿，这是为什么呢?

这是因为 500ms 的间隔我们人眼是能够感知的，而屏幕的刷新率我们是感知不到的。就相当于 500ms 的时候，div 向前移动了 1px，然后停住了，我们是感知到了，所以下次移动的时候，我们能够明显感知到卡顿。

很多小伙伴可能会问，我为什么要设置为 500ms，设置为与刷新率一致不就好了么？我们继续往下看。

2.3 setInterval 存在问题

上一节中我们手动将间隔时间改为了 500ms，虽然可能有些极端，但是在实际项目中，setInterval 本身就是有一些问题的。

影响动画的问题：

- setInterval 是异步的，也就是意味着 JS 代码执行的时候会将它放入异步队列中，所以它的执行时间并不确定。结合我们上述动画，假设我们设置的是 16.7ms 执行一次，但是在实际项目中，有可能有很多异步函数，这个时候 setInterval 执行时间隔就可能不是准确的 16.7ms，这就会造成卡顿。
- 屏幕刷新率是不定的，现在各大厂家的屏幕刷新率有 30Hz、60Hz、90Hz、120Hz 等等，以后还会更多，但是我们传入 setInterval 的时间间隔是固定的，这就有可能造成在不同设备上的动画流畅度不同。
- setInterval 会一直在后台执行，即使我们访问其它页面时。

我们做个假设，让大家更好理解问题所在：

同样是 div 从左往右移动的动画前提下，我们使用 setInterval 实现动画，且间隔设置为 10ms，屏幕的绘制过程大致如下：

1. 0ms 时：屏幕未刷新，setInterval 未执行。

2. 10ms 时：屏幕未刷新，setInterval 执行，left=1px。
3. 16.7ms 时：屏幕刷新，setInterval 未执行，left=1px。
4. 20ms 时：屏幕未刷新，setInterval 执行，left=2px。
5. 30ms 时：屏幕未刷新，setInterval 执行，left=3px。
6. 33.4ms 时：屏幕刷新，setInterval 未执行，left=3px。

依次执行.....

上面的流程中，总共费时 33.4ms，屏幕刷新了两次。正常情况下，第一次屏幕刷新时，left=1px，第二次屏幕刷新时，left 应为 2px，但是我们发现第二次屏幕刷新时，left 直接变为了 3px，这是由于我们的 setInterval 间隔设置为 10ms 的原因。这就造成了屏幕上的 div 的 left 直接从 1px 变为了 3px，这就会造成跳帧，即卡顿。

3.requestAnimationFrame 简介

前面我们介绍的 setInterval 是我们使用 JS 实现动画常用的手段，不可否认，在以前确实可以这样使用，也是最好的办法。

但是我们也需要正面这些问题，特别是随着电子设备的不断更新换代，屏幕的刷新率也出现很多种的情况下，如果继续使用定时器来实现动画，很可能会造成动画的卡顿以及性能的消耗。

有问题就有解决方案！requestAnimationFrame API 就非常完美的解决了定时器实现动画的各种问题。

我们先来看看官网如何解释它的。

官网解释：

> 告诉浏览器——你希望执行一个动画，并且要求浏览器在下次重绘之前调用指定的回调函数更新动画。该方法需要传入一个回调函数作为参数，该回调函数会在浏览器下一次重绘之前执行。
>
> 当你准备更新动画时你应该调用此方法。这将使浏览器在下一次重绘之前调用你传入给该方法的动画函数 (即你的回调函数)。

通过官网的解释我们似乎还不能很明了的知道它为什么能解决哪些问题，别急，我们一步一步来看。我们首先需要明确的是 requestAnimationFrame 是一个 API，即一个原生方法。

3.1 语法简介

我们先来看看 requestAnimationFrame 如何使用，再来感受它的魅力！

它的使用非常简单，因为它是一个原生 API。

语法如下：

```
window.requestAnimationFrame(callback);
```

参数解释：

- callback：一个回调函数，也就是我们的动画函数，就好比我们定时器接收的回调函数一般，它是下一次重绘之前更新动画帧所调用的函数 。

callback 参数我们需要重点关注它的执行时间，上面的重绘可以简单理解为重新绘制页面，动画帧也就是更新图像，也就是我们前面所说的屏幕刷新率：60Hz 的屏幕下大概每 16.7ms 就会更新依次图像，这就是动画帧。可以简单理解为回调函数会在屏幕刷新的时候调用。

返回值：

一个 long 整数，请求 ID，是回调列表中唯一的标识。是个非零值，没别的意义。你可以传这个值给 window.cancelAnimationFrame() 以取消回调函数。

3.2 特点

接下来我们来总结一下 requestAnimationFrame 的特点，知道了它的特点我们就明白它为什么厉害了！

- requestAnimationFrame 方法只接收一个参数，即回调函数，意味着我们不用间隔。因为 requestAnimationFrame 中的回调函数执行频率通常为 60 次每秒，但是它会随着屏幕刷新率的变化而变化。总之，回调函数的执行频率与浏览器的刷新次数相匹配。
- requestAnimationFrame 不会在后台一直执行，它会在页面出现的时候才会执行，比如 document.hide 为 true 的时候，而我们的定时器是一直会执行的。可以节省 CPU 和 GPU 等等。
- 可以避免丢帧，因为回调函数执行频率与浏览器刷新频率相匹配。

3.3 实际演示

我们将前面使用 setInterval 实现的动画再用 requestAnimationFrame 实现一遍。

代码如下：

```
let box = document.getElementsByClassName('box')[0];
// 动画函数
let num = 0;
function animation() {
  if (num < 1000) {
    num++;
    box.style.left = num + 'px'
    // 请求动画帧，即屏幕刷新的时候执行回调函数
    requestAnimationFrame(animation); // 继续执行该函数
  } else {
    cancelAnimationFrame(); // 清楚该函数
  }
}
```

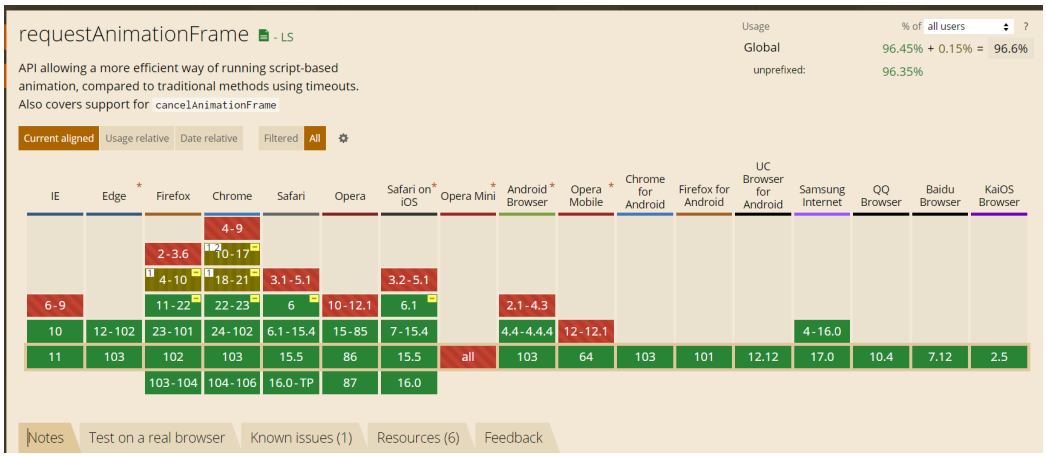
```
14
15  let raID = requestAnimationFrame(animation);
```

实现效果：

上段代码很简单，我们将动画函数传入的 requestAnimationFrame 方法中，然后在动画函数中继续调用 requestAnimationFrame 方法，以保证动画持续进行，因为我们改变了 left，是需要更新动画的。而且从上图可以看出，当我们切换到其它页面时，动画停留在了那儿，切换回来时，动画继续执行。

4.兼容性

对于这种比较新的东西，我们最好还是看看它的兼容性如何，如下图所示：



我们看到它的兼容性很不错，我们可以放心的使用。

总结

本篇文章我们用了大量篇幅来讲解动画原理，其实理由很简单，只要你搞懂了动画原理，你就明白了我们为什么要使用 requestAnimationFrame 方法，其中的道理你就自然而然的明白了！

最后，可能有些小伙伴会疑惑：为什么不用 CSS 实现上述动画呢?

确实，本篇文章的动画可以利用 CSS 实现，但是如果项目遇到无法使用 CSS 实现的动画呢，该怎么办呢?