# 深入理解requestAnimationFrame

本文从底层实现原理的角度介绍了requestAnimationFrame、cancelAnimationFrame，并给出了相关的示例代码和我对实现原理的理解和讨论。

申明

本文主要参考w3c的2013年10月31日版本的《基于脚本动画的定时控制》，进行了翻译、整理工作，并加入了自己的理解。

鉴于本人水平有限，难免理解有些偏差，欢迎大家拍砖！此处向w3c致敬！建议大家有空也看看w3c规范，里面介绍了很多底层原理，可以深入地学习Web标准。

w3c链接：

http://www.w3.org/

介绍

浏览器中的动画有两种实现形式：通过申明元素实现（如SVG中的<animate>元素）和在脚本中实现。

可以通过setTimeout和setInterval方法来在脚本中实现动画，但是这样效果可能不够流畅，且会占用额外的资源。它们有如下的特征：

1、即使向其传递毫秒为单位的参数，它们也不能达到ms的准确性。这是因为javascript是单线程的，可能会发生阻塞。

2、没有对调用动画的循环机制进行优化。

3、没有考虑到绘制动画的最佳时机，只是一味地以某个大致的事件间隔来调用循环。

其实，使用setInterval或setTimeout来实现主循环，根本错误就在于它们抽象等级不符合要求。我们想让浏览器执行的是一套可以控制各种细节的api，实现如“最优帧速率”、“选择绘制下一帧的最佳时机”等功能。但是如果使用它们的话，这些具体的细节就必须由开发者自己来完成。

本文介绍的requestAnimationFrame API不需要使用者指定循环间隔时间，而是由浏览器基于当前页面是否可见、CPU的负荷情况等来自行决定最佳的帧速率。使用该API可以让浏览器更合理地使用CPU。

本文主要内容

定义

（加入a name 链接）

**animation frame request callback list**

**动画帧请求回调函数列表**

**该列表是一个**<handle, callback> 元组，其中handle是一个整数，唯一地标识了列表的入口；callback是一个无返回值的、形参为一个时间值，该值为从1970年1月1日到当前所经过的毫秒数

刚开始列表为空。

**FrameRequestCallback** = void ([DOMHighResTimeStamp](http://www.w3.org/TR/hr-time/#domhighrestimestamp) time);

**每个Document都有一个**动画帧请求回调函数列表**，该列表可以看成是由**<handle, callback> 元组组成的集合，其中handle是一个整数，唯一地标识了元组在列表中的位置；callback是一个无返回值的、形参为一个时间值的函数，该时间值为从1970年1月1日到当前所经过的毫秒数

刚开始该列表为空。

A [Document](http://www.w3.org/TR/domcore/#document) is said to **have active**

**当Document对象有一个非空的动画帧请求回调函数列表时，我们就说该document对象有激活的动画。**

相关概念理解

Document

指Dom模型中定义的对应dom树Document的Document节点

Active document

浏览器上下文browsingContext中的Document被指定为active document。

browsingContex

浏览器上下文是呈现document对象给用户的环境。

浏览器中的1个tab或一个窗口包含一个顶级浏览器上下文，如果该页面有iframe，则iframe中也会有自己的浏览器上下文，称为嵌套的浏览器上下文。

DOM模型

详见我的《理解DOM》

document对象

当html文档加载完成后，浏览器就会创建一个document对象，它对应于Document节点，实现了HTML的Document接口。

通过该对象可获得整个html文档的信息，对 HTML 页面中的所有元素进行访问和操作。

HTML的Document接口对 DOM定义的Document接口进行了扩展，定义了 HTML 专用的属性和方法

接口定义详见[The Document object](http://www.w3.org/TR/html5/dom.html#document)。

页面可见

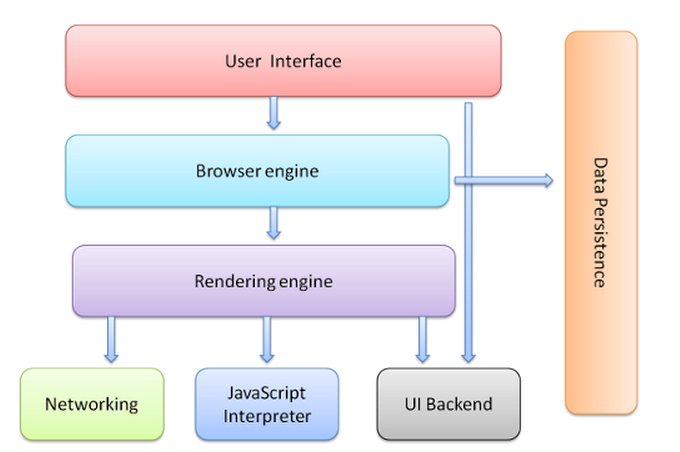
当页面被最小化或者被切换成后台标签页时,浏览器会触发一个 visibilitychange事件,并设置document.hidden属性为true；切换到显示状态的时候也同样触发一个 [visibilitychange](https://developer.mozilla.org/zh-CN/docs/Web/Reference/Events/visibilitychange)事件，设置document.hidden属性为false

详见Page Visibility <http://www.w3.org/TR/page-visibility/>

Page Visibility(页面可见性) API介绍、微拓展 <http://www.zhangxinxu.com/wordpress/2012/11/page-visibility-api-introduction-extend/>

队列

1. 大多数浏览器的组件构成如图



  在最底层的三个组件分别是网络，UI后端和js解释器。作用如下：

    （1）网络－ 用来完成网络调用，例如http请求，它具有平台无关的接口，可以在不同平台上工作

    （2）UI 后端－ 用来绘制类似组合选择框及对话框等基本组件，具有不特定于某个平台的通用接口，底层使用操作系统的用户接口

    （3）JS解释器－ 用来解释执行JS代码

 ps：上图和知识点主要来自《[HOW BROWSERS WORK: BEHIND THE SCENES OF MODERN WEB BROWSERS](http://www.html5rocks.com/en/tutorials/internals/howbrowserswork/)》 想深入了解的同学可以重点看下。

      2. 大多数浏览器让一个单线程（如chrome的render进程中的render线程）共用于执行javascrip和更新用户界面。这个线程通常被称为“浏览器UI线程”， 每个时刻只能执行其中一种操作，这意味着当Javascript代码正在执行时用户界面无法响应输入，反之亦然。这样做是因为javascript代码的作用就是操作DOM更新用户界面，用同一个线程来做负责这两件事情可以更高效

      3. 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，包括重绘和重排。

API接口

In the definition of[requestAnimationFrame](http://www.w3.org/TR/animation-timing/#dom-windowanimationtiming-requestanimationframe) below, references to the[Document](http://www.w3.org/TR/domcore/#document) object are to be taken to be references to the [Window](http://www.w3.org/TR/html5/browsers.html#the-window-object) object’s [active document](http://www.w3.org/TR/html5/browsers.html#active-document). [[HTML5]](http://www.w3.org/TR/animation-timing/#ref-HTML5)

Window对象定义了以下两个接口：

partial interface **Window** {

long [requestAnimationFrame](http://www.w3.org/TR/animation-timing/#dom-windowanimationtiming-requestanimationframe)([FrameRequestCallback](http://www.w3.org/TR/animation-timing/#framerequestcallback) callback);

void [cancelAnimationFrame](http://www.w3.org/TR/animation-timing/#dom-windowanimationtiming-cancelanimationframe)(long handle);

};

[requestAnimationFrame](http://www.w3.org/TR/animation-timing/#dom-windowanimationtiming-requestanimationframe)

[requestAnimationFrame](http://www.w3.org/TR/animation-timing/#dom-windowanimationtiming-requestanimationframe)方法用于通知浏览器重采样动画调用动画的回调函数。

当requestAnimationFrame(callback)被调用时，浏览器会将动画重采样的回调函数调用任务插入到**动画帧请求回调函数列表的末尾，并且返回一个handle值，该值为浏览器定义的、大于0的整数，唯一标识了该回调函数在列表中位置。**

每个回调函数都有一个布尔标识cancelled，该标识初始值为false，并且对外不可见。

这里可以理解为requestAnimationFrame(callback)被调用时并不会执行callback，而是会将一个元组<handle, callback>插入到动画帧请求回调函数列表末尾，返回该元组的handle值，其中元组的callback就是传入requestAnimationFrame的回调函数。浏览器会在定期加入到主线程中的采样所有动画的任务中遍历动画帧请求回调函数列表，判断每个元组的callback的cancelled，如果为false，则执行callback。详见后面的“处理模型”。

**cancelAnimationFrame**

**cancelAnimationFrame** 用来取消先前安排的一个动画帧更新的请求。当调用cancelAnimationFrame(handle)时，浏览器会设置该handle指向的回调函数的cancelled为true。无论该回调函数是否在**动画帧请求回调函数列表中，它的cancelled都会被设置为true。如果该handle没有指向任何回调函数，则调用该函数不会发生任何事情。**

**注意：**

[cancelAnimationFrame](http://www.w3.org/TR/animation-timing/#dom-windowanimationtiming-cancelanimationframe) might be called for an entry in the[Document](http://www.w3.org/TR/domcore/#document)’s [animation frame request callback list](http://www.w3.org/TR/animation-timing/#dfn-animation-frame-request-callback-list) or in the [sample all animations](http://www.w3.org/TR/animation-timing/#dfn-sample-all-animations)operation’s temporary list. In either case the entry’s cancelled flag is set to true so that the callback does not run.

处理模型

当页面可见并且**动画帧请求回调函数列表不为空时，浏览器会定期地加入一个采样所有动画的任务到队列（**UI线程**）中（单线程），该任务对应的是document的顶级上下文（），**

用伪代码叙述

**采样所有动画的任务的执行如下所示：**

// var list = 空的动画帧请求回调函数列表;

var list = {};

//这里的browsingContexts指页面的顶级浏览器上下文和其iframe（如果有iframe的话）的嵌套浏览器上下文

var browsingContexts = 浏览器顶级上下文及其下属的浏览器上下文;

for (var browsingContext in browsingContexts) {

var time = 从1970年1月1日到当前所经过的毫秒数;

var d = browsingContext的active document;

if (d.hidden !== true) { //如果该active document可见

//拷贝d的动画帧请求回调函数列表到list中，并清空动画帧请求回调函数列表

var doclist = d的动画帧请求回调函数列表

doclist.appendTo(list);

clear(doclist);

}

//遍历动画帧请求回调函数列表的元组中的回调函数

for (var callback in list) {

if (callback.cancelled !== true) {

try {

//每个browsingContext都有一个对应的WindowProxy对象，WindowProxy对象会将callback指向active document关联的window对象。

callback.call(window, time);

}

catch (e) { //忽略异常

}

}

}

}

给出代码进行讲解

假设。。。

Html

讨论

为什么在callback内部调用cancelAnimationFrame不能取消动画，在callback外部调用则可以取消？

如下面的代码会一直执行a：

var id = null;

function a(time) {

console.log("animation");

window.cancelAnimationFrame(id);

id = window.requestAnimationFrame(a);

}

a();

我们来分析下这段代码是如何执行的：

执行“a();”，执行函数a；

执行“console.log("animation");”，打印“animation”；

执行“window.cancelAnimationFrame(id);”，因为id为null且动画帧请求回调函数列表为空，浏览器在动画帧请求回调函数列表中找不到对应的callback，所以不发生任何事情；

执行“id = window.requestAnimationFrame(a);”，浏览器会将一个元组<handle, callback>插入到Document的动画帧请求回调函数列表末尾，将id赋值为该元组的handle值，指向callback。其中元组的callback就是回调函数a；

a执行完毕。

因为当前页面可见（假设代码执行时页面可见）并且**动画帧请求回调函数列表不为空，**浏览器会定期地加入一个采样所有动画的任务到线程队列中，所以当a执行完毕后，会有“采样所有动画”的任务执行：

拷贝Document的动画帧请求回调函数列表到list变量中，清空Document的动画帧请求回调函数列表；

遍历list的列表，列表有1个元组，该元组的callback为a；

判断a的cancelled，为默认值false，所以执行a；

执行“console.log("animation");”，打印“animation”；

执行“window.cancelAnimationFrame(id);”，此时id指向当前元组的a（即当前正在执行的a），浏览器将当前元组的a的cancelled设为true。

执行“id = window.requestAnimationFrame(a);”，浏览器会将新的元组<handle, callback>插入到Document的动画帧请求回调函数列表末尾（新元组的a的cancelled为默认值false！），将id赋值为该元组的handle值。

当下一个“采样所有动画”的任务执行时，会判断动画帧请求回调函数列表的元组的a的cancelled，因为a为执行“id = window.requestAnimationFrame(a);”时新插入的元组的callback，所以该值为默认值false，因此会继续执行a！

如果代码为：

var id = null;

function a(time) {

console.log("animation");

id = window.requestAnimationFrame(a);

window.cancelAnimationFrame(id);

}

a();

或者为

function a(time) {

console.log("animation");

id = window.requestAnimationFrame(a);

}

a();

window.cancelAnimationFrame(id);

则只会执行一次a。

因为“window.cancelAnimationFrame(id);”的id指向新插入到动画帧请求回调函数列表中的元组的a，因此在下一次的“采样所有动画”任务中的判断元组的a的cancelled时，值为true，因此不执行a。

如果在引擎钩子方法中调用window.cancelAnimationFrame ，则取消的\_loopId指向的是本次主循环。然而本次主循环已经开始了，因此主循环不会停止

例如用户在Scene->onstartLoop中调用Director->stop，试图取消本次主循环(\_loopId指向的本次循环)，然而本次主循环已经开始了，因此主循环不会停止。只有在本次主循环中取消下一次主循环，但是该钩子方法是在Director->\_loopBody中调用的，而\_loopBody执行完后，会返回Director->\_run中执行“this.\_loopId = window.requestNextAnimationFrame(function (time) {”，这时loopId又被赋予新的值，开始下一次主循环，而之前调用的“window.cancelAnimationFrame (this.\_loopId);”取消的是上一次的

如果用户在引擎主循环外部调用，则可以停止主循环。

（requestAnimationFrame是将回调函数插入到列表中，也就是说回调函数并不是立即执行！那什么时候执行？）

注意：

1、因为在“采样所有动画”的任务中会清空动画帧请求回调函数列表，所以如果要多次调用回调函数，则requestAnimationFrame每次只安排一次回调函数调用动画重采样的任务（因为），如果需要多次采样动画，则需要在回调函数中再次调用requestAnimationFrame。

例如下面的代码只执行1次animate函数：

var id = null;

function animate(time) {

console.log("animation");

}

window.requestAnimationFrame(animate);

下面代码会一直执行animate函数：

var id = null;

function animate(time) {

console.log("animation");

window.requestAnimationFrame(animate);

}

//与执行“animate();”不同，执行“window.requestAnimationFrame(animate);”时并不会立即执行animate，

//而是会将callback为animate的元组插入到Document的动画帧请求回调函数列表末尾，在下一次的“采样所有动画”任务中再执行。

window.requestAnimationFrame(animate);

2、如果在回调函数被调用或者列表被清空之前多次调用requestAnimationFrame插入同一个回调函数，那么列表中就会有多个元组指向该回调函数（它们的handle不同，但callback都指向该回调函数），因此该回调函数就会被执行多次。

例如下面的代码， animate会将两个元组（handle分别为id1、id2，callback都为animate）插入到Document的动画帧请求回调函数列表末尾。因为在“采样所有动画”的任务中会遍历执行动画帧请求回调函数列表的每一个元组，所以执行“采样所有动画”的任务时都会执行两次animate。

var id1 = null,

id2 = null;

function animate(time) {

console.log("animation");

id1 = window.requestAnimationFrame(animate);

id2 = window.requestAnimationFrame(animate);

}

animate();

window.cancelAnimationFrame (id1);

window.cancelAnimationFrame (id2);

3、（requestAnimationFrame是将回调函数插入到列表中，也就是说回调函数并不是立即执行！那什么时候执行？）

兼容性 polyfill方法

/\*\*

\* 出自《HTML5 Canvas 核心技术》

\*/

window.requestNextAnimationFrame = (function () {

var originalWebkitRequestAnimationFrame = undefined,

wrapper = undefined,

callback = undefined,

geckoVersion = 0,

userAgent = navigator.userAgent,

index = 0,

self = this;

// Workaround for Chrome 10 bug where Chrome

// does not pass the time to the animation function

if (window.webkitRequestAnimationFrame) {

// Define the wrapper

wrapper = function (time) {

if (time === undefined) {

time = +new Date();

}

self.callback(time);

};

// Make the switch

originalWebkitRequestAnimationFrame = window.webkitRequestAnimationFrame;

window.webkitRequestAnimationFrame = function (callback, element) {

self.callback = callback;

// Browser calls the wrapper and wrapper calls the callback

originalWebkitRequestAnimationFrame(wrapper, element);

}

}

// Workaround for Gecko 2.0, which has a bug in

// mozRequestAnimationFrame() that restricts animations

// to 30-40 fps.

if (window.mozRequestAnimationFrame) {

// Check the Gecko version. Gecko is used by browsers

// other than Firefox. Gecko 2.0 corresponds to

// Firefox 4.0.

index = userAgent.indexOf('rv:');

if (userAgent.indexOf('Gecko') != -1) {

geckoVersion = userAgent.substr(index + 3, 3);

if (geckoVersion === '2.0') {

// Forces the return statement to fall through

// to the setTimeout() function.

window.mozRequestAnimationFrame = undefined;

}

}

}

return window.requestAnimationFrame ||

window.webkitRequestAnimationFrame ||

window.mozRequestAnimationFrame ||

window.oRequestAnimationFrame ||

window.msRequestAnimationFrame ||

function (callback, element) {

var start,

finish;

window.setTimeout(function () {

start = +new Date();

callback(start);

finish = +new Date();

self.timeout = 1000 / 60 - (finish - start);

}, self.timeout);

};

}());

window.cancelNextRequestAnimationFrame = window.cancelRequestAnimationFrame

|| window.webkitCancelAnimationFrame

|| window.webkitCancelRequestAnimationFrame

|| window.mozCancelRequestAnimationFrame

|| window.oCancelRequestAnimationFrame

|| window.msCancelRequestAnimationFrame

|| clearTimeout;

参考资料

[Timing control for script-based animations](http://www.w3.org/TR/animation-timing/)

[Browsing contexts](http://www.w3.org/TR/html5/browsers.html#windows)

[The Document object](http://www.w3.org/TR/html5/dom.html#document)

引擎工作原理（名称还要修改）

《HTML5 Canvas核心技术》

《理解DOM》

Page Visibility <http://www.w3.org/TR/page-visibility/>

《[HOW BROWSERS WORK: BEHIND THE SCENES OF MODERN WEB BROWSERS](http://www.html5rocks.com/en/tutorials/internals/howbrowserswork/)》