

17 | 如何使用Canvas绘制统计图表（上）？

2019-08-22 李超

从0打造音视频直播系统

[进入课程 >](#)



讲述：李超

时长 17:37 大小 12.10M



Canvas 是 HTML5 标准中的一个新元素，你可以把它想像成一块“画布”，有了它你就可以在网页上绘制图像和动画了。在 HTML5 页面中可像使用其他元素一样使用 Canvas，如 Video 标签。为了能够在 Canvas 上绘图，浏览器为此提供了一整套 JavaScript API，我们将在后面的代码中看到如何使用它们。

实际上，早期要想在浏览器的网页上实现动画效果，需要用到 Flash 技术。但通过 Flash 实现动画非常繁琐，你必须专门去学习 Flash 相关的知识。而 Canvas 则完全不用，它就是一个简单的标签，通过几行 JavaScript 代码你就可以很容易地控制它。

Canvas 最早由苹果公司开发的，用在 Mac OS X 的 WebKit 组件中。之后，各大主流的浏览器都对 Canvas 进行了支持，所以它也被纳入到了 HTML5 的规范中。

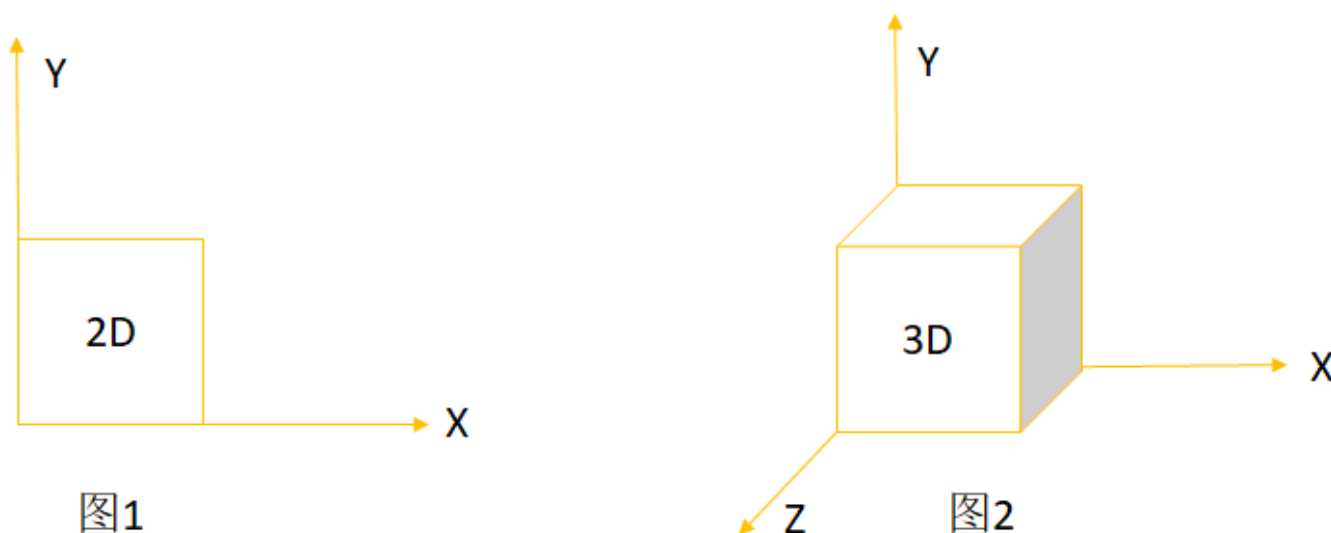
Canvas 支持 2D 和 3D 图像的绘制，并且应用领域非常广泛，基本上涵盖了 Web 图形 / 图像、视频、动画等领域。我们在网页中经常见到的统计图表、视频应用、网页游戏等等，都是由它来实现的。

基本概念和原理

在深入讲解 Canvas 之前，我们先来了解一些基础概念和原理。对于这些基础概念的理解，将会为你学习后面的知识点打下坚实的基础。

1. 2D 图像与 3D 图像

在真实世界中，我们看到的都是三维世界（3 Dimension），即立体图像。而在计算机系统中，显示的图像都是二维的，它是一个平面，有**水平**和**垂直**两个维度。下面这张图是 2D 与 3D 的对比图。



2D 与 3D 对比图

在计算机中，3D 图像是在 2D 图像的基础上通过增加一个**深度**来实现的，就像上图中的图 2 所示。当然，由于咱们的显示器是二维的，所以三维图像最终显示的时候要将 3D 图像转换成 2D 图像，然后才能在显示器上显示出来，这与像机拍照是同样的原理。

2. 矢量图

所谓矢量图，实际上是一些“基本图元 + 计算公式”，图像是在展示时通过实时计算后**绘制出来的**。非矢量图是一张像素的二维数组，图像就是由这个二维数组中的**每个像素拼出来**的。

举几个简单的例子你就更清楚了：

如果用矢量图表述一条线，那它实际上只需要存储两个点（起点和终点）的坐标就可以了。而对于非矢量图，则需要将这两点之间的所有像素都要绘制出来。

如果用矢量图表述一个圆，则它只记录一个圆点和圆的半径即可。对于非矢量图，它会将圆上的每一个像素绘制出来。

通过上面两个例子，我想你已经很清楚矢量图与非矢量图之间的区别了。常见的 BITMAP、PNG、JPEG 这些类型的图形都属于非矢量图，它们都是通过绘制像素点构成的图形。也可以这么说，基本上我们日常生活中的图像都是非矢量图。

3. 图形 / 图像渲染的基本原理

在讲解图形 / 图像渲染原理之前，你需要先知道什么是分辨率，实际上**分辨率**的概念非常简单，就是像素的宽高值，我想你应该是非常清楚了。

分辨率也是显示器的一个重要指标，现在显示器的**分辨率**越来越高，以前能达到 1080P 就觉得非常牛了（且价格不菲），但现在 2K 屏已经成为主流，甚至 4K 屏也不是特别稀奇的了。

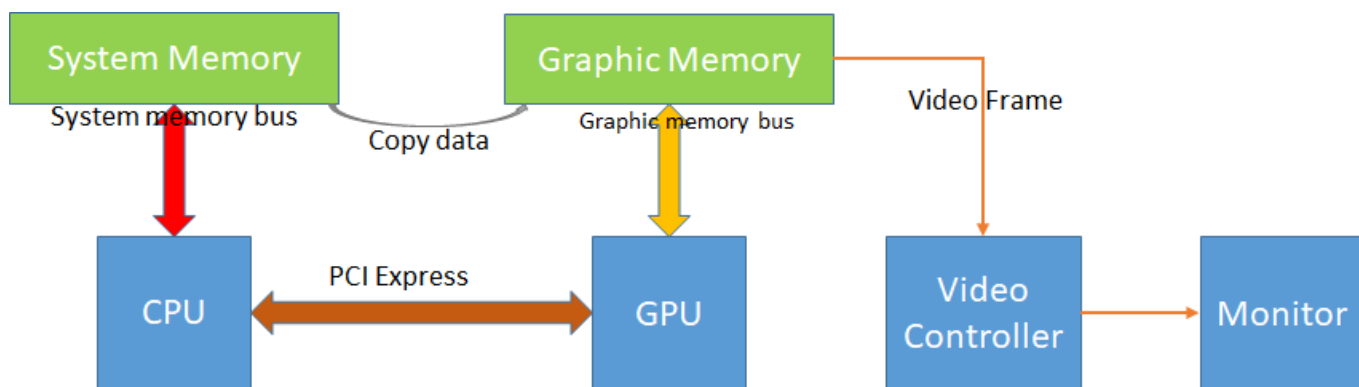
比如，我的显示器的分辨率是 $1920 * 1080$ ，这里的 1920 和 1080 指的是显示器有 1920 列和 1080 行，也就是说我的显示器是由 $1920 * 1080$ 个像素组成的阵列。每一个像素由 R、G、B 三种颜色组成，将这样成千上万的像素连接起来，就形成了一幅图像。

如果你想将一幅图在屏幕上显示出来，那么只需要将图形转换成像素图，这样就可以通过连接起来的像素将图像展示在显示器上。以上这个过程就是图像渲染的过程，它有一个专业术语，叫做**光栅化**。

除了上面讲到的光栅化这个术语之外，你可能还会经常看到“软渲染”“硬件加速”等术语。实际上，它们指的是由谁进行光栅化，是 CPU，还是 GPU？所谓“**软渲染**”是通过 CPU 将图形数据光栅化；而“**硬件加速**”则是指通过 GPU 进行光栅化处理。

其实，在 GPU 出现之前，图形处理都是通过 CPU 完成的。只不过随着计算机图形图像技术在各个行业的广泛应用，尤其是 3D 图形处理需要大量的计算，所以产生了 GPU 芯片，专门用于图形处理，以加快图像渲染的性能，并将 CPU 释放出来。

下面我们看一下，现代计算机图形处理的基本原理，如下图所示：



图形处理原理图

图中 Graphic Memory、GPU、Video Controller 都是现代显卡的关键组成部分，具体处理过程大致可描述为如下。

应用程序处理的图形数据都是保存在 System Memory 中的，也就是我们经常所说的主内存中。需要硬件处理的时候，先将 System Memory 中的图形数据拷贝到 Graphic Memory 中。

然后，通过 CPU 指令通知 GPU 去处理图形数据。

GPU 收到指令后，从 Graphic Memory 读取图形数据，然后进行坐标变换、着色等一系列复杂的运算后形成像素图，也就是 Video Frame，会存储在缓冲区中。

视频控制器从 Video Frame 缓冲区中获取 Video Frame，并最终显示在显示器上。

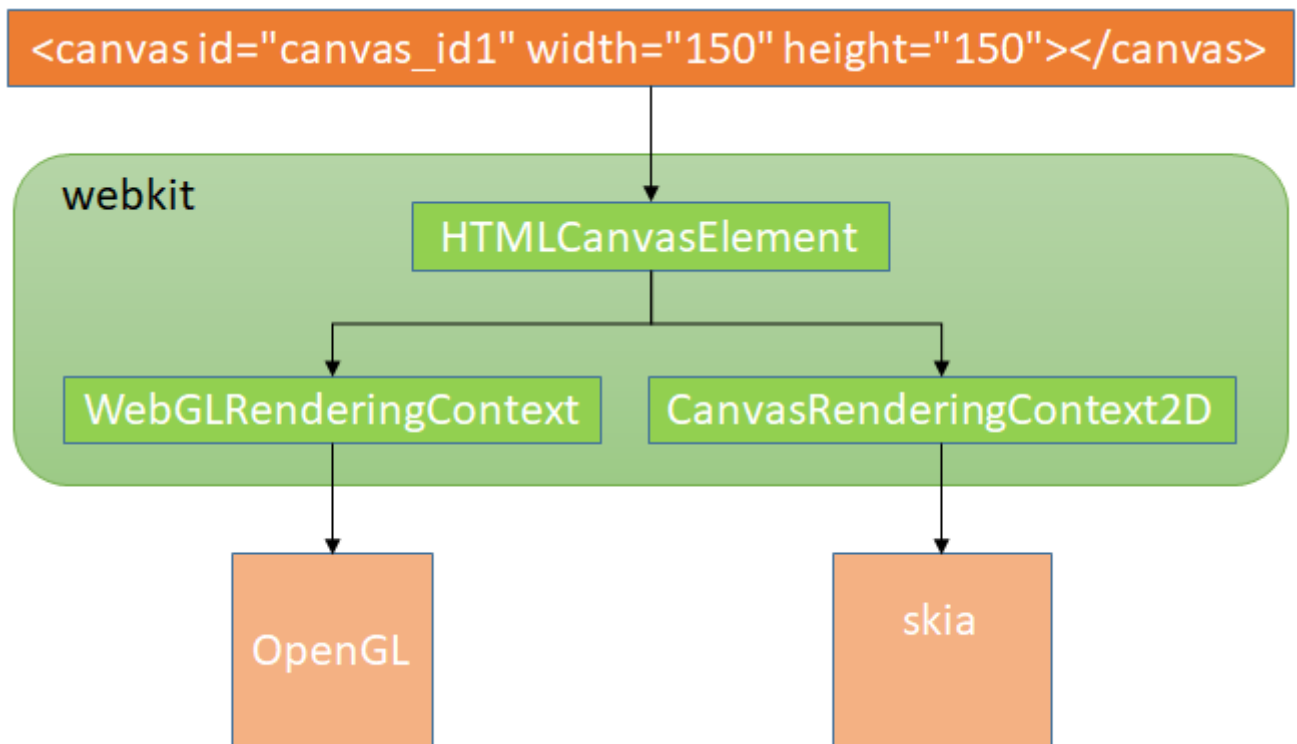
4. Canvas 实现原理

目前苹果浏览器使用的渲染引擎是 WebKit，Chrome 使用的渲染引擎是基于 WebKit 修改后的 Blink，它们的基本原理都差不多，只是实现细节的差别。我们这里以 WebKit 为例，来看看它是如何进行图像渲染的。

无论是 WebKit 还是 Blink，它们都是通过 Canvas 元素对图像进行渲染。（对 WebKit 有兴趣的同学可以从 GitHub 上获取到其源码，GitHub 地址为：

<https://github.com/WebKit/webkit/tree/master/Source/WebCore/html/canvas>)

下面我们来看一下 HTML 中的 Canvas 在 WebKit 里是如何表述的，其实现原理图如下所示：



Canvas 实现原理图

在 WebKit 中，实现 Canvas 的逻辑还是非常复杂的，不过通过上面的原理图你可以了解到，它是由以下几个模块组成：

CanvasRenderingContext2D 类，用于对 2D 图形渲染。

WebGLRenderingContext 类，用于对 3D 图形渲染。

2D 图形渲染的底层使用了 Google 开源的 Skia 库。

3D 图形渲染的底层使用的是 OpenGL，不过在 Windows 上使用的却是 D3D。

5. Skia 库

Skia 库最早是由 Skia 公司开发，2005 年被 Google 收购。Google 于 2007 年将其开源。**Skia 库是开源的 2D 图形库**，由 C++ 语言实现，目前用在 Chrome、Android、Firefox 等产品。

Skia 支持基于 CPU 的软渲染，同时也支持基于 OpenGL 等库的硬件加速。这个库性能比较高，而且方便使用，可以大大提高开发者的效率。

你目前只需要对这个库有一些简单的了解就行，如果在你的工作中需要对视频 / 图像进行渲染，再对其进行更详细的了解。

6. OpenGL VS WebGL

OpenGL (Open Graphic Library) 是一个应用编程接口，用于 2D/3D 图形处理。最早是由 Silicon Graphics 公司提出的，目前是由 Khronos Group 维护。


OpenGL 虽说是一个 API，但其实并没有代码实现，只是定义了接口规范。至于具体实现则是由各个显卡厂家提供的。

OpenGL ES 规范是基于 OpenGL 规范的，主要用于嵌入式平台。目前 Android 系统上的视频 / 图像渲染都是使用的 OpenGL ES。当然，以前 iOS 也是使用 OpenGL ES 做视频 / 图像渲染，不过现在已经改为使用自家的 Metal 了。

WebGL 是用于在 HTML 页面上绘制 3D 图像规范。它是基于 OpenGL 规范的。比如，WebGL 1.0 规范是基于 OpenGL 2.0 规范，WebGL 2.0 规范基于 OpenGL 3.0 规范。

Canvas 的使用

HTML5 新增了 <Canvas> 元素，你把它想像成一块画布就可以了，其使用方式如下：

 复制代码

```
1 <canvas id="tutorial" width="300" height="150"></canvas>
```

在 HTML 中像上面一样增加 Canvas 元素后，就可以在其上面绘制各种图形了。在 Canvas 中有 width 和 height 两个重要的属性，你可以通过它们设置 Canvas 的大小。


当然，你也可以不给 Canvas 设置任何宽高值，在没有给 Canvas 指定宽高值的情况下，width 和 height 的默认值是 300 和 150 像素。

另外，Canvas 元素本身并没有绘制功能，但你可以通过 Canvas 获取到它的渲染上下文，然后再通过 Canvas 的渲染上下文，你就可以使用底层的 OpenGL 或 Skia 来进行视频 / 图像的渲染了。

1. Canvas 渲染上下文 (Render Context)

在浏览器内部，Canvas 是使用状态机进行管理的。几乎 Canvas 每个属性的改变都会导致其内部的状态发生变化，而各种视频 / 图形的绘制都是在不同的状态下进行的。所以你若要在 Canvas 上绘制图形，就首先要知道 Canvas 的状态，而这个状态如何获取到呢？

Canvas 提供了一个非常方便的方法，即 **getContext 方法**。也就是说，Canvas 的状态机是在它的上下文中存放着的。那么，现在来看一下 getContext API 的格式吧，其格式如下：

 复制代码

```
1 var ctx = canvas.getContext(contextType);
2 var ctx = canvas.getContext(contextType, contextAttributes);
```

其中，参数 contextType 是必填参数，其类型是一个 **DOMString** 类型的字符串，取值有如下：

2d，将会返回 CanvasRenderingContext2D 类型的对象实例，这是一个 2D 图像渲染上下文对象，底层使用 Skia 渲染库。


webgl，返回 WebGLRenderingContext 类型的对象实例，这是一个 3D 图像渲染上下文对象，使用 WebGL 1.0 规范，是基于 OpenGL 2.0 规范的。

webgl2，返回 WebGL2RenderingContext 类型的对象实例，也是一个 3D 图像渲染上下文对象，使用 WebGL 2.0 规范，是基于 OpenGL 3.0 规范的。

bitmaprenderer，返回 ImageBitmapRenderingContext 类型的对象实例。

而参数 contextAttributes 是可选的，一般情况下不使用，除非你想使用一些高级特性。

下面我们就来看一下具体的例子，来看看如何通过 Canvas 元素绘制一些图形，代码如下：

 复制代码

```
1 ...
2 let canvas = document.getElementById('canvas_id1'); // 从 HTML 获取到 Canvas
3 let ctx_2d = canvas.getContext('2d'); // 得到 Canvas 的渲染上下文
4
5 ctx_2d.fillStyle = "rgb(200,0,0)"; // 设置颜色为红色
6 ctx_2d.fillRect (10, 10, 55, 50); // 设置矩形的大小
7
```

```
8 ctx_2d.fillStyle = "rgba(0, 0, 200, 0.5)"; // 设置颜色为蓝色，并且透明
9 ctx_2d.fillRect (30, 30, 55, 50); // 设置矩型大小
10 ...
```

在上面代码中，首先通过 `getElementById` 获得了 `Canvas` 元素，然后通过 `Canvas` 元素获得了它的渲染上下文。因为在获取上下文时传入的参数为**2d**，所以可以知道是获取的 2D 图像渲染上下文。拿到上下文后，通过上下文的 `fillRect` 方法就可以将图形绘制出来了。


下面是这段代码最终的运行结果图：



2. 浏览器兼容性适配

由于 `Canvas` 是从 `HTML5` 之后才开始有的，所以比较老的浏览器是不支持 `<Canvas>` 标签的。不过，各浏览器在 `Canvas` 兼容性方面做得还是非常不错的。

对于支持 `Canvas` 的浏览器，直接会渲染画布中的图像或者动画；对于不支持 `Canvas` 的浏览器来说，你可以在 `<Canvas>` 和 `<Canvas>` 中间添加替换的内容，这样浏览器会直接显示替换的内容。替换的内容可以是文字描述，也可以是静态图片。比如下面的代码：

 复制代码

```
1 <canvas id="canvas_id1" width="150" height="150">
2   The browser doesn't support the canvas tag.
3 </canvas>
```


它表示在 IE8 执行上面的 HTML 页面，就会显示 The browser doesn't support the canvas tag. 这句提示信息。

总结

图形图像相关的应用非常广泛，涉及的概念很多，技术也很复杂。在本文的开头，我向你讲解了 2D、3D 图形的基本概念和图形渲染的基本原理；后面简要地分析了 Canvas 在 WebKit 中的实现原理，并介绍了 Canvas 支持 3D 图形绘制所遵循的标准；最后，又说明了 Canvas 元素及其使用方法。

当然还有很多图形学的概念、原理在本文中并没有被提及或分析，比如，坐标系统、RGB 颜色模型、GPU 架构等等，由于篇幅原因，这里咱们就不做过多介绍了。如果你对这方面感兴趣的话，可以自行搜索相关内容并进一步学习。

至于 Canvas 在 WebRTC 中的具体应用，我会在下篇中再向你做详细介绍。

思考时间

今天你要思考的问题是：HTML5 中的视频播放除了可以使用 Video 标签播放之外，你还能想到其他可以使用的实现方式吗？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

从 0 打造音视频直播系统

手把手教你打造实时互动音视频直播系统

李超

新东方音视频直播技术专家
前沪江音视频架构师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 16 | WebRTC中的数据统计原来这么强大（下）

下一篇 18 | 如何使用Canvas绘制统计图表（下）？

精选留言 (2)

写留言



许童童

2019-08-22

思考题：

可以通过canvas来解析渲染二进制流，根据视频的编码方式，选择相应的解码方式即可。
这一节讲的内容还是挺基础的，不过正好复习一下。



2



周伟民

2019-08-22

思考题回答：（1）H.264视频解码后通常是YUV420P格式，可以把YUV转成RGBA，调用Canvas的2D上下文接口来逐帧显示视频。（2）或者调用Canvas的webgl上下文，直接渲染YUV数据，也可显示视频。参考案例：<https://github.com/v354412101/wsPlayer>

展开

作者回复: 赞！

