

01 | 浏览器中实现可视化的四种方式

2020-06-22 月影

跟月影学可视化

[进入课程 >](#)



讲述：月影

时长 18:35 大小 17.02M



你好，我是月影。

上一节课我们了解了什么是可视化。可视化用一句话来说，本质上就是将数据信息组织起来后，以图形的方式呈现出来。在 Web 上，图形通常是通过浏览器绘制的。现代浏览器是一个复杂的系统，其中负责绘制图形的部分是渲染引擎。渲染引擎绘制图形的方式，我总结了一下，大体上有 4 种。

第 1 种是传统的 **HTML+CSS**。这种方式通常用来呈现普通的 Web 网页。



第 2 种是使用 **SVG**。SVG 和传统的 **HTML+CSS** 的绘图方式差别不大。只不过，HTML 元素在绘制矢量图形方面的能力有些不足（我们后面会讲到），而 SVG 恰好弥补了这方面的缺陷。

第 3 种是使用 **Canvas2D**。这是浏览器提供的 Canvas API 中的其中一种上下文，使用它可以非常方便地绘制出基础的几何图形。在可视化中，Canvas 比较常用，下一节课我们会学习它的基本用法。

第 4 种是使用 **WebGL**。这是浏览器提供的 Canvas API 中的另一种上下文，它是 OpenGL ES 规范在 Web 端的实现。我们可以通过它，用 GPU 渲染各种复杂的 2D 和 3D 图形。值得一提的是，WebGL 利用了 GPU 并行处理的特性，这让它在处理大量数据展现的时候，性能大大优于前 3 种绘图方式。因此，在可视化的应用中，一些数据量大、视觉效果要求高的特殊场景，使用 WebGL 渲染是一种比较合适的选择。

这 4 种方式各有利弊，今天我就从宏观层面带你了解这些图形系统，为我们后面更深入的学习打好基础。

方式一：HTML+CSS

与传统的 Web 应用相比，可视化项目，尤其是 PC 端的可视化大屏展现，不只是使用 HTML 与 CSS 相对较少，而且使用方式也不太一样。于是，有些同学就会认为，可视化只能使用 SVG、Canvas 这些方式，不能使用 HTML 与 CSS。当然了，这个想法是不对。具体的原因是什么呢？我一起来看看。

实际上，现代浏览器的 HTML、CSS 表现能力很强大，完全可以实现常规的图表展现，比如，我们常见的柱状图、饼图和折线图。

虽然我们后面的课程会主要使用 Canvas 和 WebGL 绘图，少数会涉及部分 CSS。但是，你可不要觉得它不重要。为啥呢？理由有两个：

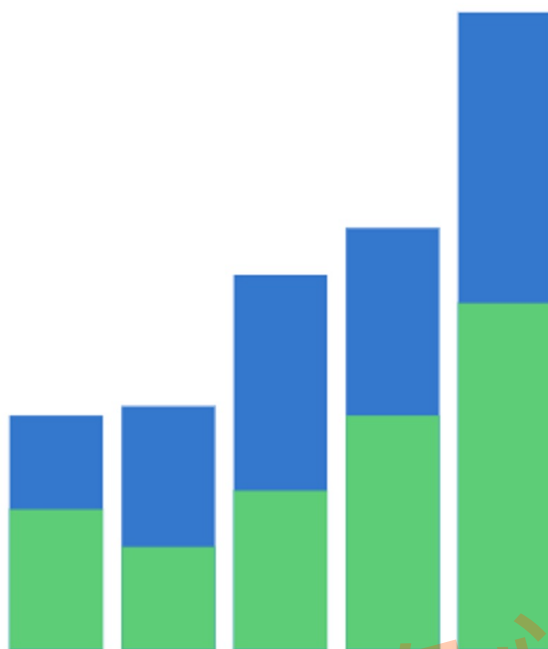
一些简单的可视化图表，用 CSS 来实现很有好处，既能简化开发，又不需要引入额外的库，可以节省资源，提高网页打开的速度。

理解 CSS 的绘图思想对于可视化也是很有帮助的，比如，CSS 的很多理论就和视觉相关，可视化中都可以拿来借鉴。

所以呢，这一节里我们多讲一点，你一定要好好听。接下来，我们就来说一说，CSS 是如何实现常规图表的。

1. HTML 与 CSS 是如何实现可视化的？

用 CSS 实现柱状图其实很简单，原理就是使用网格布局（Grid Layout）加上线性渐变（Linear-gradient），我就不多说了，你可以直接看我这里给出的 CSS 代码。



用HTML+CSS绘制的柱状图

复制代码

```
1 /**
2  dataset = {
3    current: [15, 11, 17, 25, 37],
4    total: [25, 26, 40, 45, 68],
5  }
6  */
7  .bargraph {
8    display: grid;
9    width: 150px;
10   height: 100px;
11   padding: 10px;
12   transform: scaleY(3);
13   grid-template-columns: repeat(5, 20%);
14 }
15 .bargraph div {
16   margin: 0 2px;
17 }
18 .bargraph div:nth-child(1) {
19   background: linear-gradient(to bottom, transparent 75%, #37c 0, #37c 85%, #3c
20 }
21 .bargraph div:nth-child(2) {
22   background: linear-gradient(to bottom, transparent 74%, #37c 0, #37c 89%, #3c
23 }
24 .bargraph div:nth-child(3) {
```

```

25 background: linear-gradient(to bottom, transparent 60%, #37c 0, #37c 83%, #3c
26 }
27 .bargraph div:nth-child(4) {
28 background: linear-gradient(to bottom, transparent 55%, #37c 0, #37c 75%, #3c
29 }
30 .bargraph div:nth-child(5) {
31 background: linear-gradient(to bottom, transparent 32%, #37c 0, #37c 63%, #3c
32 }

```

而要实现饼图，我们可以使用圆锥渐变，方法也很简单，你直接看代码就可以理解。




使用圆锥渐变绘制的饼图

```

1 .piegraph {
2   display: inline-block;
3   width: 250px;
4   height: 250px;
5   border-radius: 50%;
6   background-image: conic-gradient(#37c 30deg, #3c7 30deg, #3c7 65deg, orange 65deg, #37c 65deg);
7 }

```

 复制代码

柱状图和饼图都比较简单，所以我带你快速过了一下。除此之外，我们用 HTML 和 CSS 也可以实现折线图。

我们可以用高度很小的 Div 元素来模拟线段，然后用 transform 改变角度和位置，这样就能拼成折线图了。另外，如果使用 clip-path 这样的高级属性，我们还能实现更复杂的图表，比如，用不同的颜色表示两个不同折线的面积。



折线图和面积图

实际上很多常见的可视化图表我们都可以用 HTML 和 CSS 来实现，不需要用其他的绘图方式。但是，为什么在可视化领域很少有人直接用 HTML 和 CSS 来绘制图表呢？这主要是因为，使用 HTML 和 CSS 绘图，有 2 个缺点。

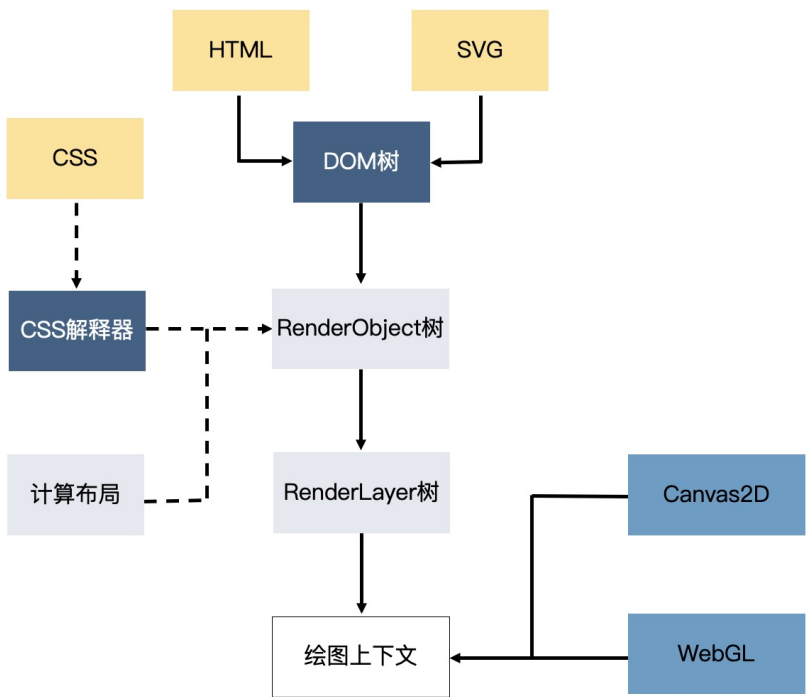
2. 用 HTML+CSS 实现可视化的缺点

首先，HTML 和 CSS 主要还是为网页布局而创造的，使用它们虽然能绘制可视化图表，但是绘制的方式并不简洁。这是因为，从 CSS 代码里，我们很难看出数据与图形的对应关系，有很多换算也需要开发人员自己来做。这样一来，一旦图表或数据发生改动，就需要我们重新计算，维护起来会很麻烦。

其次，HTML 和 CSS 作为浏览器渲染引擎的一部分，为了完成页面渲染的工作，除了绘制图形外，还要做很多额外的工作。比如说，浏览器的渲染引擎在工作时，要先解析 HTML、SVG、CSS，构建 DOM 树、RenderObject 树和 RenderLayer 树，然后用 HTML（或 SVG）绘图。当图形发生变化时，我们很可能要重新执行全部的工作，这样的性能开销是非常大的。

而且传统的 Web 开发，因为涉及 UI 构建和内容组织，所以这些额外的解析和构建工作都是必须做的。而可视化与传统网页不同，它不太需要复杂的布局，更多的工作是在绘图和数据计算。所以，对于可视化来说，这些额外的工作反而相当于白白消耗了性能。

因此，相比于 HTML 和 CSS，Canvas2D 和 WebGL 更适合去做可视化这一领域的绘图工作。它们的绘图 API 能够直接操作绘图上下文，一般不涉及引擎的其他部分，在重绘图像时，也不会发生重新解析文档和构建结构的过程，开销要小很多。



图形系统与浏览器渲染引擎工作对比

方式二：SVG

在介绍 Canvas2D 和 WebGL 之前，我们先来说一说 SVG。现代浏览器支持 SVG (Scalable Vector Graphics, 可缩放矢量图)，SVG 是一种基于 XML 语法的图像格式，可以用图片 (img 元素) 的 src 属性加载。而且，浏览器更强大的是，它还可以内嵌 SVG 标签，并且像操作普通的 HTML 元素一样，利用 DOM API 操作 SVG 元素。甚至，CSS 也可以作用于内嵌的 SVG 元素。

比如，上面的柱状图，如果用 SVG 实现的话，我们可以用如下所示的代码来实现：

复制代码

```
1 <!--
2     dataset = {
```



```

3      total: [25, 26, 40, 45, 68],
4      current: [15, 11, 17, 25, 37],
5  }
6  -->
7  <svg xmlns="http://www.w3.org/2000/svg" width="120px" height="240px" viewBox="
8    <g transform="translate(0, 100) scale(1, -1)">
9      <g>
10         <rect x="1" y="0" width="10" height="25" fill="#37c"/>
11         <rect x="13" y="0" width="10" height="26" fill="#37c"/>
12         <rect x="25" y="0" width="10" height="40" fill="#37c"/>
13         <rect x="37" y="0" width="10" height="45" fill="#37c"/>
14         <rect x="49" y="0" width="10" height="68" fill="#37c"/>
15      </g>
16      <g>
17         <rect x="1" y="0" width="10" height="15" fill="#3c7"/>
18         <rect x="13" y="0" width="10" height="11" fill="#3c7"/>
19         <rect x="25" y="0" width="10" height="17" fill="#3c7"/>
20         <rect x="37" y="0" width="10" height="25" fill="#3c7"/>
21         <rect x="49" y="0" width="10" height="37" fill="#3c7"/>
22      </g>
23    </g>
24  </svg>

```

从上面的 SVG 代码中，我们可以一目了然地看出，数据 total 和 current 分别对应 SVG 中两个 g 元素下的 rect 元素的高度。也就是说，元素的属性和数值可以直接对应起来。而 CSS 代码并不能直观体现出数据的数值，需要进行 CSS 规则转换。具体如下图所示：

```

dataset = {
  total: [25, 26, 40, 45, 68],
  current: [15, 11, 17, 25, 37],
}

.bargraph div:nth-child(1) {
  background: linear-gradient(to bottom, transparent 75%, #37c 0, #37c 85%, #3c7 0);
}

<rect x="1" y="0" width="10" height="25" fill="#37c"/>
<rect x="13" y="0" width="10" height="26" fill="#37c"/>
<rect x="25" y="0" width="10" height="40" fill="#37c"/>

```

在上面这段 SVG 代码中，g 表示分组，rect 表示绘制一个矩形元素。除了 rect 外，SVG 还提供了丰富的图形元素，可以绘制矩形、圆弧、椭圆、多边形和贝塞尔曲线等等。由于 SVG 比较复杂，我们会在第 4 节课专门介绍，如何用 SVG 绘制可视化图表。在那之前，你也可以通过 [MDN 官方文档](#)，来学习更多的 SVG 的 API。

SVG 绘制图表与 HTML 和 CSS 绘制图表的方式差别不大，只不过是将 HTML 标签替换成 SVG 标签，运用了一些 SVG 支持的特殊属性。

HTML 的不足之处在于 HTML 元素的形状一般是矩形，虽然用 CSS 辅助，也能够绘制出各种其它形状的图形，甚至不规则图形，但是总体而言还是非常麻烦的。而 SVG 则弥补了这方面的不足，让不规则图形的绘制变得更简单了。因此，用 SVG 绘图比用 HTML 和 CSS 要便利得多。

但是，SVG 图表也有缺点。在渲染引擎中，SVG 元素和 HTML 元素一样，在输出图形前都需要经过引擎的解析、布局计算和渲染树生成。而且，一个 SVG 元素只表示一种基本图形，如果展示的数据很复杂，生成图形的 SVG 元素就会很多。这样一来，大量的 SVG 元素不仅会占用很多内存空间，还会增加引擎、布局计算和渲染树生成的开销，降低性能，减慢渲染速度。这也就注定了 SVG 只适合应用于元素较少的简单可视化场景。

方式三：Canvas2D

除了 SVG，使用 Canvas2D 上下文来绘制可视化图表也很方便，但是在绘制方式上，Canvas2D 和 HTML/CSS、SVG 又有些不同。

无论是使用 HTML/CSS 还是 SVG，它们都属于**声明式**绘图系统，也就是我们根据数据创建各种不同的图形元素（或者 CSS 规则），然后利用浏览器渲染引擎解析它们并渲染出来。但是 Canvas2D 不同，它是浏览器提供的一种可以直接用代码在一块平面的“画布”上绘制图形的 API，使用它来绘图更像是传统的“编写代码”，简单来说就是调用绘图指令，然后引擎直接在页面上绘制图形。这是一种**指令式**的绘图系统。

那 Canvas 到底是怎么绘制可视化图表的呢？我们一起来看。

首先，Canvas 元素在浏览器上创建一个空白的画布，通过提供渲染上下文，赋予我们绘制内容的能力。然后，我们只需要调用渲染上下文，设置各种属性，然后调用绘图指令完成输出，就能在画布上呈现各种各样的图形了。

为了实现更加复杂的效果，Canvas 还提供了非常丰富的设置和绘图 API，我们可以通过操作上下文，来改变填充和描边颜色，对画布进行几何变换，调用各种绘图指令，然后将绘制的图形输出到画布上。

总结来说，Canvas 能够直接操作绘图上下文，不需要经过 HTML、CSS 解析、构建渲染树、布局等一系列操作。因此单纯绘图的话，Canvas 比 HTML/CSS 和 SVG 要快得多。

但是，因为 HTML 和 SVG 一个元素对应一个基本图形，所以我们可以很方便地操作它们，比如在柱状图的某个柱子上注册点击事件。而同样的功能在 Canvas 上就比较难实现了，因为对于 Canvas 来说，绘制整个柱状图的过程就是一系列指令的执行过程，其中并没有区分“A 柱子”、“B 柱子”，这让我们很难单独对 Canvas 绘图的局部进行控制。不过，这并不代表我们就不能控制 Canvas 的局部了。实际上，通过数学计算我们是可以通过定位的方式来获取局部图形的，在后续的课程中我们会解决这个问题。

这里有一点需要你注意，Canvas 和 SVG 的使用也不是非此即彼的，它们可以结合使用。因为 SVG 作为一种图形格式，也可以作为 image 元素绘制到 Canvas 中。举个例子，我们可以先使用 SVG 生成某些图形，然后用 Canvas 来渲染。这样，我们就既可以享受 SVG 的便利性，又可以享受 Canvas 的高性能了。

方式四：WebGL

WebGL 绘制比前三种方式要复杂一些，因为 WebGL 是基于 OpenGL ES 规范的浏览器实现的，API 相对更底层，使用起来不如前三种那么简单直接。关于 WebGL 的使用内容，我会在 3D 篇详细来说。

一般情况下，Canvas2D 绘制图形的性能已经足够高了，但是在三种情况下我们有必要直接操作更强大的 GPU 来实现绘图。

第一种情况，如果我们**要绘制的图形数量非常多**，比如有多达数万个几何图形需要绘制，而且它们的位置和方向都在不停地变化，那我们即使用 Canvas2D 绘制了，性能还是会达到瓶颈。这个时候，我们就需要使用 GPU 能力，直接用 WebGL 来绘制。

第二种情况，如果我们**对较大图像的细节做像素处理**，比如，实现物体的光影、流体效果和一些复杂的像素滤镜。由于这些效果往往要精准地改变一个图像全局或局部区域的所有像

素点，要计算的像素点数量非常的多（一般是数十万甚至上百万数量级的）。这时，即使采用 Canvas2D 操作，也会达到性能瓶颈，所以我们也要用 WebGL 来绘制。

第三种情况是**绘制 3D 物体**。因为 WebGL 内置了对 3D 物体的投影、深度检测等特性，所以用它来渲染 3D 物体就不需要我们对坐标做底层的处理了。那在这种情况下，WebGL 无论是在使用上还是性能上都有很大优势。

要点总结

今天，我们介绍了四种可视化实现方式和它们的优缺点。

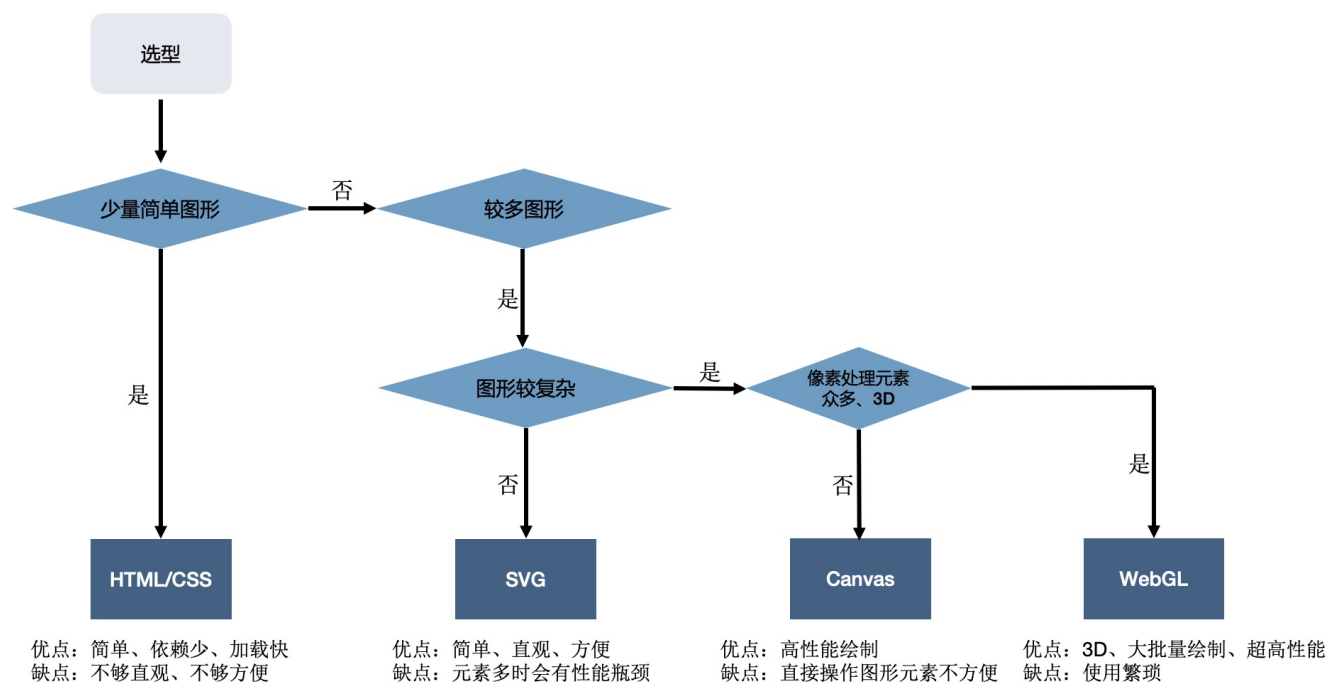
HTML+CSS 的优点是方便，不需要第三方依赖，甚至不需要 JavaScript 代码。如果我们要绘制少量常见的图表，可以直接采用 HTML 和 CSS。它的缺点是 CSS 属性不能直观体现数据，绘制起来也相对麻烦，图形复杂会导致 HTML 元素多，而消耗性能。

SVG 是对 HTML/CSS 的增强，弥补了 HTML 绘制不规则图形的能力。它通过属性设置图形，可以直观地体现数据，使用起来非常方便。但是 SVG 也有和 HTML/CSS 同样的问题，图形复杂时需要的 SVG 元素太多，也非常消耗性能。

Canvas2D 是浏览器提供的简便快捷的指令式图形系统，它通过一些简单的指令就能快速绘制出复杂的图形。由于它直接操作绘图上下文，因此没有 HTML/CSS 和 SVG 绘图因为元素多导致消耗性能的问题，性能要比前两者快得多。但是如果绘制的图形太多，或者处理大量的像素计算时，Canvas2D 依然会遇到性能瓶颈。

WebGL 是浏览器提供的功能强大的绘图系统，它使用比较复杂，但是功能强大，能够充分利用 GPU 并行计算的能力，来快速、精准地操作图像的像素，在同一时间完成数十万或数百万次计算。另外，它还内置了对 3D 物体的投影、深度检测等处理，这让它更适合绘制 3D 场景。

知道了这些优缺点，在实际面对可视化需求的时候，我们就可以根据具体项目的特点来选择合适的方案实现可视化了。那具体来说，我们应该怎么选择呢？我这里梳理了一张技术方案的选择图，你可以看一看。



小试牛刀

我们在文中实现了 SVG 版本的柱状图，你可以尝试用 SVG 实现同 HTML/CSS 版本一样的饼图、折线图和面积图，体会一下使用 SVG 实现和 HTML/CSS 实现的不同点。

另外，下一节课我们会介绍 Canvas2D 绘制可视化图表，你可以提前预习一下，试一试能否用 Canvas2D 来绘制文中的柱状图。

欢迎在留言区和我讨论，分享你的答案和思考，也欢迎你把这节课分享给你的朋友，我们下节课见！

跟月影学可视化

系统掌握图形学与可视化核心原理

月影

奇虎 360 奇舞团团长

可视化 UI 框架 SpriteJS 核心开发者



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 预习 | Web前端与可视化到底有什么区别？

下一篇 02 | 指令式绘图系统：如何用Canvas绘制层次关系图？

精选留言 (14)

 写留言



寻找海蓝

2020-06-22

所以 Canvas 实际上是比较 svg 更底层的API：

1. svg 本质上是 Dom，是一种具有特殊属性的 Dom，Dom的优势和缺点它都具备，我们虽然可以像操作Dom那样方便地操作Canvas，但是不得不面对Dom性能低下的事实，所以 svg 只适合低动画需求、少元素数量需求的场景。...

展开 ∨

 3

 4



Light Yagami

2020-06-22

月影大佬，我来报道了

展开 ∨



Meow戴

2020-06-23

canvas2d绘制出来的图形最终也是渲染到gpu中的吧，和webgl渲染到底区别在哪里，为啥webgl性能好啊，请问月影团长~~

展开 ∨



一步

2020-06-23

Canvas 2D 和 WebGL 都是 Canvas API 的一种上下文，那么 Canvas API 到底有多少中上下文 API 呢？

还有问题就是 Canvas API 是不是就是一个接口规范，Canvas 2D 和 WebGL 只是其中的一种实现？

展开 ∨



TYY

2020-06-22

老师讲的非常有条理了

展开 ∨

作者回复: 谢谢~



gltjk

2020-06-24

按老师的建议用 <path> 重构了 SVG 饼图（不过由声明式变成了命令式😂），同时把数据与视图分离了。<https://codepen.io/gltjk/pen/vYLmdvJ>

展开 ∨



潘pan

2020-06-23

月影爸爸~~ 我来报道了

展开 ∨





浮生若梦

2020-06-23

希望老师后面讲具体技术的时候多结合项目的难点、前沿趋势、底层原理讲解，不要讲成入门demo



浮生若梦

2020-06-23

canvas和webgl都是GPU绘图，但是他们在代码中都是以js的形式来写的。那么首先需要解析js然后才能绘制，这里是CPU在起作用。但是后期更改图像（js代码）的时候，还是纯GPU操作吗？会不会插入js的CPU操作，最后绘制才进入GPU操作？相比html和css只是少了前期布局和绘制的步骤，因为浏览器最后将html和css的图像展现在页面最终也是会到GPU里。另外，threejs中的shader貌似比一般的threejs（如精灵）效果更好，这又是什...
展开 ∨



啊啊啊黑猫警长

2020-06-23

图形中运用到的数学知识,除了后面要讲的以外.平时应该通过什么来学习积累呢.典型案例吗



王子晨

2020-06-23

大佬请问，svg和canvas的混合使用，对于性能消耗怎样的，可以理解为这种混合模式在性能上优于canvas，低于webGL么？

展开 ∨



.a_.c.

2020-06-23

老师期待继续的课程

顺便问问有计划出书不？[调皮]

展开 ∨



gltjk

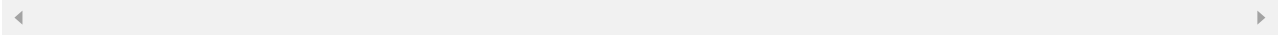
2020-06-23

用 SVG 做了文章里 HTML+CSS 实现的饼图，感觉麻烦多了，要自己计算极坐标.....http

s://codepen.io/gltjk/pen/XWXRdVr

展开 ▾

作者回复: 棒! 看了一下代码, 写得挺好的。不过SVG用circle和stroke-dasharray来做, 一般来说双色的饼图这样做简单, 多色的感觉直接用path来做是不是会简单些呢?



💬 1



彭涛 Chris

2020-06-23

所以 Canvas 是怎么实现的呢? 也是用webgl吗? dom节点的渲染最终也是webgl吗?

作者回复: 不是用WebGL, 但是肯定用了GPU渲染。

