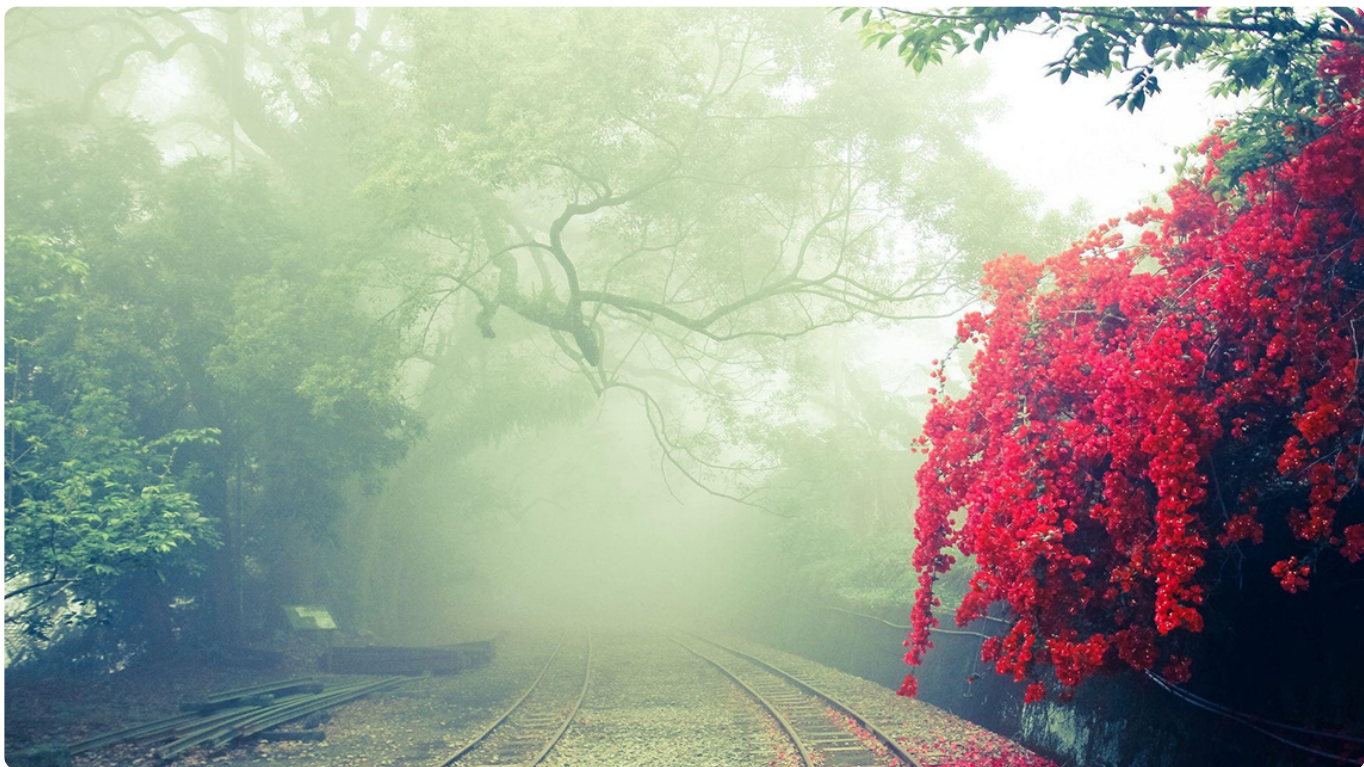


## 18 | 千言万语不及一幅画：谈谈数据可视化

2019-10-21 四火

全栈工程师修炼指南

[进入课程 >](#)



**讲述：四火**

时长 21:37 大小 14.86M



你好，我是四火。

随着大数据和数据分析趋势的流行，数据可视化变得越来越重要，而许多全栈的学习材料并没有跟上节奏，去介绍这方面的技术。这一讲中，我们将介绍数据可视化的基本概念和原理，以及几个常用的 JavaScript 用来实现数据可视化的库。

数据可视化，即 Data Visualization，是指使用具备视觉表现力的图形和表格等工具，向用户传达数据信息的方式。在我工作过的每个大型团队中，数据可视化技术都有着其不可替代的用武之地。

在大数据分析团队，数据库可视化技术被用来分析数据变化，验证机器学习算法的效果；在高可用服务团队，数据可视化技术被用来了解和监视服务的运行状况，了解系统的压力和负

载；在分布式平台团队，数据可视化技术用来俯瞰一个个异步任务的执行情况，以获知任务执行的健康状况.....事实上，只要有工程的地方，数据可视化就扮演着举足轻重的角色。

## Web 绘图标准

在前端绘图，是数据可视化里面很常见的一个需求，我们常见的有位图和矢量图这样两种。

通常我们谈论的图片，绝大多数都是位图。位图又叫栅格图像，**无论位图采用何种压缩算法，它本质就是点阵**，它对于图像本身更具备普适性，无论图像的形状如何，都可以很容易分解为一个二维的点阵，更大的图，或者更高的分辨率，只是需要更密集的点阵而已。

你可能已经听说过矢量图。**矢量图是使用点、线段或者多边形等基于数学方程的几何形状来表示的图像**。将一个复杂图像使用矢量的方式来表达，显然要比位图困难得多，但是矢量图可以无损放大，因为它的本质是一组基于数学方程的几何形状的集合，因此无论放大多少倍，形状都不会发生失真或扭曲。并且图像越大，就越能比相应的位图节约空间，因为矢量图的大小和实际图像大小无关。倘若再采用独立的压缩算法进行压缩，矢量图可以基于文本压缩，从而获得很大的压缩比。

在早些年的项目中，在后端使用 Python 等语言预生成绘制图像的场景还比较多，但是如今已经少见一些了，大多数的图形生成都被搬到了前端。而这种情况也成为了前后端分离，以及数据和展示分离的典型场景，后端同步或异步生成不同维度的数据，浏览器则通过统一的 API 根据用户需求获取相应的数据；前端根据这些取得的数据在浏览器中现场绘制图像。关于服务端和客户端聚合的知识，如有遗忘，请回看 [\[第 09 讲\]](#)。

总的来看，前端绘图，和后端比起来，有这样几个显著的优势。

**前端生成的图形图像具有天然的交互性。**前端生成的图像不仅仅意味着一张“图”，还意味着它能够和 HTML 这样的呈现结构紧密地结合起来，而图像上的组成部分都可以响应用户的行为。

**图像的生成可能需要显著的资源消耗，放到前端可以减轻服务器压力。**这里的消耗既包括 CPU、内存等物理资源消耗，还有用户的等待时间消耗，在前端可以更好地给用户提供渲染过程的反馈。

**图形图像的设计和规划本就属于呈现层，系统架构上把它放到前端更容易实现前后端分离，组织结构上能让擅长视觉处理的前端工程师和 UX 设计师更自然地工作。**有了数

据，就可以对前端的图像生成逻辑进行设计和测试，工程师和设计师只需要专注于前端的通用技能就可以较为独立地完成工作。


我们较常听到的 Web 绘图标准包括 VML、SVG 和 Canvas，其中 VML 是微软最初参与制定的标准，一直以来只有 IE 等少数浏览器支持，从 2012 年的 IE 10 开始它逐渐被废弃了；但是剩余两个，SVG 和 Canvas 有一定互补性，且如今都非常流行，下面我来介绍一下。

## 1. SVG

SVG 即 Scalable Vector Graphics，可缩放矢量图形。它是基于可扩展标记语言 (XML)，用于描述二维矢量图形的一种图形格式。在它之前，微软曾经向 W3C 交过 VML 的提议，但被拒绝了。之后才有了 SVG，由 W3C 制定，是一个开放标准，当时在 W3C 自己看来，SVG 的竞争对手应该主要是 Flash。

SVG 格式和前面提到的 VML 一样，支持脚本，容易被搜索引擎索引。SVG 可以嵌入外部对象，比如文字、PNG、JPG，也可以嵌入外部的 SVG。它在移动设备上存在两个子版本，分别叫做 SVG Basic 和 SVG Tiny。SVG 很快获得了各种浏览器的支持，一开始 IE 还坚守着自家的 VML 不放，但后来也慢慢被迫转移到了 SVG 的阵营，从 IE 9 才开始对 SVG 部分支持。

SVG 支持三种格式的图形：矢量图形、栅格图像和文本。所以你看，**SVG 并不只是一个矢量图的简单表示规范，而是尝试把矢量图、位图和文字统一起来的标准**。我们来亲自写一个 SVG 的小例子，在你的工作文件夹中建立 example.svg，并用文本编辑器打开，录入如下文字：

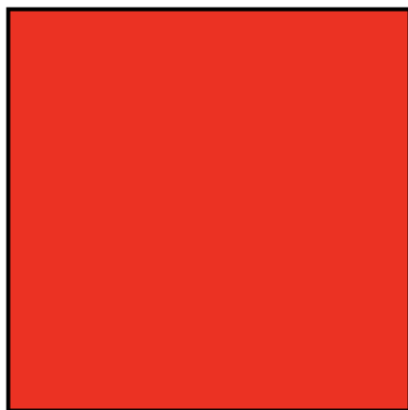
 复制代码

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="300" height="300">
3   <rect x="60" y="60" width="200" height="200" fill="red" stroke="black" stroke-width="1">
4 </svg>
```


我来对上述 XML 做个简单的解释：第一行指明了 XML 的版本和编码；第二行是一个 svg 的根节点，指明了协议和版本号，图像画布的大小（500 x 500），其中只包含一个矩形

(rect)，这个矩形的起始位置是 (x, y)，宽和高都为 200，填充红色，并使用 2px 宽的黑色线条来描边。

接着使用 Chrome 来打开这个文件，你将看到这样的效果：



接着我们另建立一个 HTML 文件：svg.html，加上 html 标签，并拷贝 XML 中的 svg 标签到这个 HTML 文件中：


 复制代码

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4   <svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="300" height="300">
5     <rect x="60" y="60" width="200" height="200" fill="red" stroke="black" stroke-width:
6   </svg>
7 </body>
8 </html>
```

用 Chrome 打开看看效果——嗯，再次展示了这个红色方块。反复点击 Chrome 的 View 菜单下的 Zoom In 选项，将图像放到最大，观察矩形的边角，没有任何模糊和失真，这证明了它确实是矢量图。



最后打开 Chrome 的开发者工具，在控制台键入：

 复制代码


```
1 $("svg>rect").setAttribute("fill", "green");
```

你会看到这个矢量图从红色变成了绿色。这充分说明，svg 就是普普通通的 HTML 标签，它可以响应 JavaScript 的控制。自此，图像对于天天和 HTML 打交道的程序员来说，再也不是一个“二进制黑盒”了。

## 2. Canvas

Canvas 标签是 HTML 5 的标签之一，标签可以定义一片区域，允许 JavaScript 动态渲染图像。开始由苹果推出，自家的 Safari 率先支持，IE 从 IE 9 开始支持。

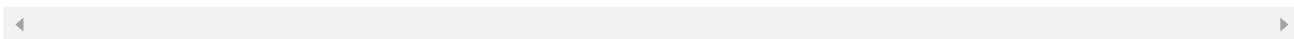
Canvas 和 SVG 有相当程度的互补之处，我们来实现一个 Canvas 的例子，体会下这一点。请在任何工作文件夹中，建立 canvas.html，并写入：

 复制代码

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4   <canvas width="300" height="300"></canvas>
5   <script type="text/javascript">
6     var canvas = document.getElementsByTagName('canvas')[0];
7     var ctx = canvas.getContext('2d');
8     ctx.rect(60,60,200,200);
9
10    ctx.fillStyle = 'RED';
11    ctx.fill();
12
13    ctx.strokeStyle = 'BLACK';
14    ctx.stroke();
```



```
15 </script>
16 </body>
17 </html>
```



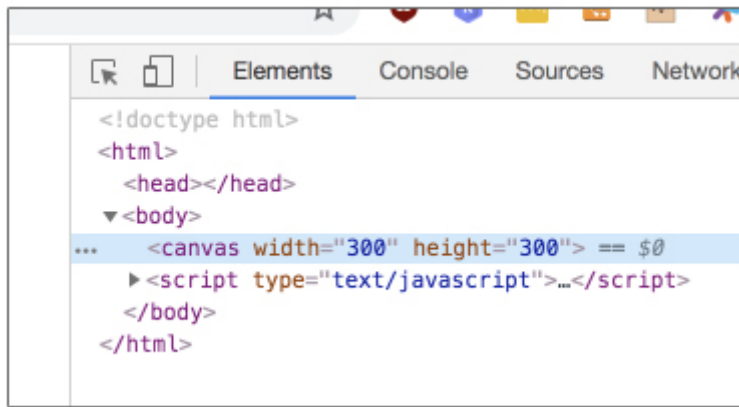
代码很容易理解，获取到 canvas 节点以后，获取一个 2D 上下文，接着设置好矩形的位置和大小，分别进行填充和描线的操作。接着使用 Chrome 打开，你会发现效果和 SVG 的例子一样，展示了这个具备黑色边框的红色方块。



看起来和 SVG 差不多对不对？我们也来执行相同的操作，反复点击 Chrome 的 View 菜单下的 Zoom In 选项，将图像放到最大：



你看，矩形的边角不再清晰，这说明这种方式绘制的不是矢量图，而是位图。再使用 Chrome 的开发者工具，点击左上角的 DOM 选择箭头，选中这个矩形，我们发现，和 SVG 不同的是，这个 canvas 节点内部并没有任何 DOM 结构。



虽然这是一个小小的例子，但足以看出 Canvas 和 SVG 之间的明显差异和互补性了。

总的来说，从图片描述过程上来说，SVG 是 HTML 标签原生支持的，因此就可以使用这种 **声明式的语言**来描述图片，它更加直观、形象、具体，每一个图形组成的 DOM 都可以很方便地绑定和用户交互的事件。**这种在渲染技术上通过提供一套完整的图像绘制模型来实现的方式叫做 Retained Mode。**

Canvas 则是藉由 JavaScript 的**命令式的语言**对既定 API 的调用，来完成图像的绘制，canvas 标签的内部，并没有任何 DOM 结构，这让它无法使用传统的 DOM 对象绑定的方式来和图像内部的元素进行互动，但它更直接、可编程性强，在浏览器内存中不需要为了图形维护一棵巨大的 DOM 树，这也让它在遇到大量的密集对象时，拥有更高的渲染性能。**这种在渲染技术上通过直接调用图形对象的绘制命令接口来实现的方式叫做 Immediate Mode。**

讲到这里，不知道你是否联想到了我们之前反复提到过的，声明式编程和命令式编程在全栈技术中的应用，如果你忘记了，可以回看 [\[第 07 讲\]](#) 中的介绍。所以，我想再次说，技术都是相通的。

Canvas	SVG
Pixel based (Dynamic .png)	Shape based
Single HTML element	Multiple graphical elements, which become part of the DOM
Modified through script only	Modified through script and CSS
Event model/user interaction is granular (x,y)	Event model/user interaction is abstracted (rect, path)
Performance is better with smaller surface, a larger number of objects (>10k), or both	Performance is better with smaller number of objects (<10k), a larger surface, or both

(上图来自 [SVG vs canvas: how to choose](#), 比较了 SVG 和 Canvas 其中的一些优劣)

我们从这些例子中可以看出，无论选用哪一种技术，HTML 5 的出现，都给了浏览器底气。以往由于其自身能力的限制，浏览器的很多领土都被播放器控件、Flash 等蚕食了，HTML 5 正助其将领土重新夺回来（你可能已经听说了，Chrome 已经开始用提示条警告：从 2020 年 12 月起 Chrome 将不再支持 Flash）。

使用这种方式，以往浏览器内的这些插件和扩展的“黑盒”全部通过原生的 HTML 标签完成替换支持，少了一个软件“层”，多了一分透明，视频、音频等媒体由浏览器底层直接支持，性能会更加出色，交互性更好。

## 数据可视化的 JavaScript 库

数据可视化的 JavaScript 库有很多，我想它们可以简单分为两类：绝大多数都比较专精，完成某一类的图表绘制工作，比如 [Flot](#)；但是也有一些相对通用而强大，比如 [D3.js](#)。

### 1. Flot


Flot 是一个非常简单的图表绘制的 jQuery 插件，这样类似的库有很多，它们绝大多数包含这样两个特点：

在使用上都包含 DOM 选择、选项设置、数据绑定、行为绑定等几个常见步骤，简单、直接，没有特定的领域语言，也没有复杂的模式套用；

它们往往针对性解决特定的、狭窄领域的问题，比如就是用来绘制二维坐标图，或者就是用来生成二维表格。

我们拿 Flot 举例，来感受一下这两个特点，比如下面这个例子，绘制一条正弦曲线，代码非常得简洁。


首先在 HTML 页面中建立一个 div：

 复制代码

```
1 <div id="plot"></div>
```



接着在 JavaScript 中，写入如下代码：

 复制代码

```
1 let data = $.map([...Array(1000).keys()],
2                 (x, i) => [[i, Math.sin(x/100)]]);
3 $.plot("#plot", [{ data }], {
4   xaxis: { ticks: [
5     0,
6     [ 100 * Math.PI, "Pi" ],
7     [ 200 * Math.PI, "2Pi" ],
8     [ 300 * Math.PI, "3Pi" ]
9   ]}
10 });
```

按照前面说的常见步骤，我来简单解释一下。

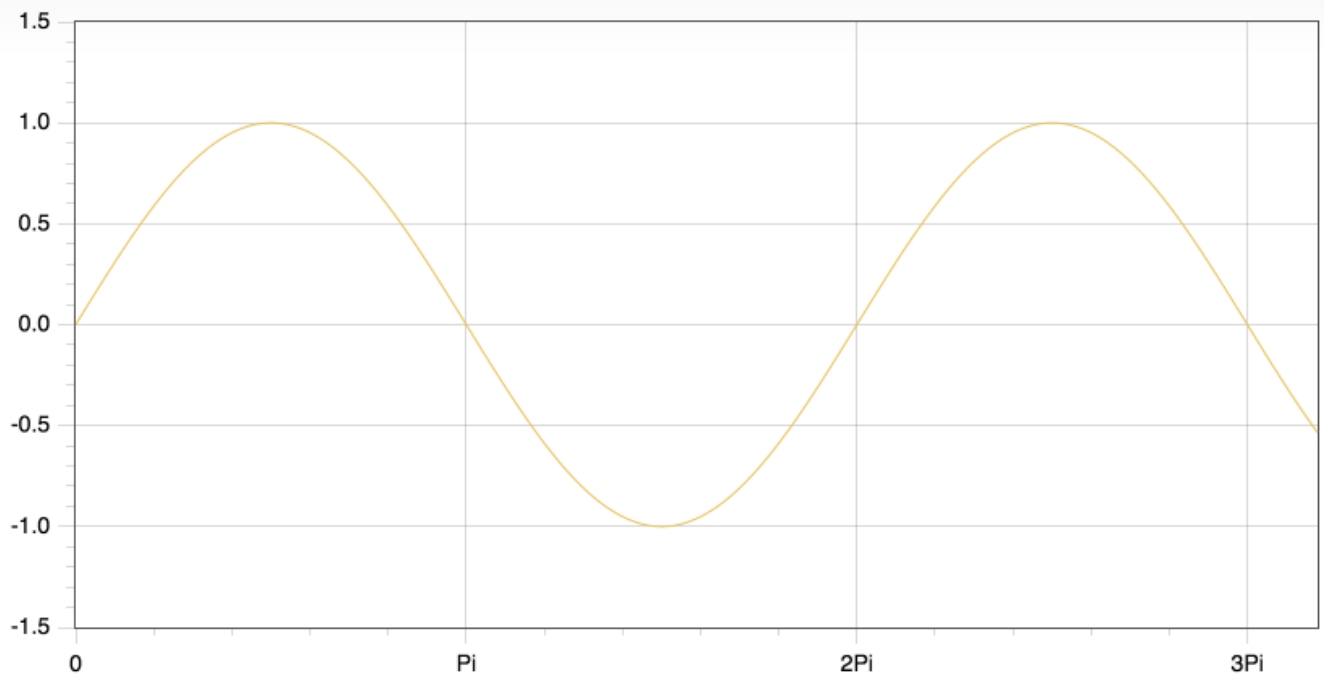
DOM 选择：“#plot” 是 jQuery 的选择器，取得了 id 为 plot 的 DOM；

数据绑定：从 0 到 1000 的数中，给每一个数除以 100，再取它的正弦，将结果和数的序号捆绑起来放到入参 data 中；

行为绑定：这里没有显示绑定行为，有一些默认的响应行为由库实现；

选项设置：后面跟着的参数，其中包含 xaxis 用于设置 x 轴的坐标显示。

通过这样简单的代码，就可以得到如下效果：



如果你考察生成的对象，你会发现它是使用 Canvas 来绘制的。

## 2. D3.js

第二类可视化 JavaScript 库相对较为通用。D3.js 是一个基于数据的操作文档的 JavaScript 库，可以让你绑定任何数据到 DOM，支持 DIV 这类常规 DOM 进行的图案生成，也支持 SVG 这种图案的生成。D3 帮助你屏蔽了浏览器差异，并且**通过基于容器和数据匹配状态变更的解耦设计，这种方式对于绘制某些动态变化的、画布元素根据数据按照一定规则变动的图像，代码会非常得清晰简洁。**

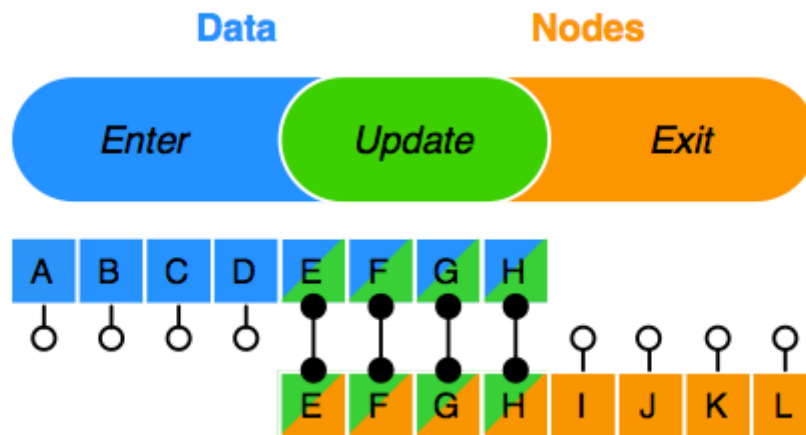
这种方法就是 “Enter and Exit” 机制，下面我们来着重理解一下它。

这种机制建立在容器节点和数据映射的关系上，即 “一个萝卜一个坑”，数据项就是萝卜，容器节点就是坑。在数据变动的过程中，通过每个节点位置和每个数据项的匹配，发生如下三种行为之一：

如果数据项能够找到它所属的节点，发生 update 事件；

如果数据项更多，节点数量不够，对于无法找到节点的数据项，发生 enter 事件；

如果数据项减少，即原有的数据项离开了节点，发生 exit 事件。



(来自 [D3: Data-Driven Documents](#))

下面，我们还是让代码说话，用一个简单的小例子，来展示这个过程。HTML 中有这样一个 DOM，作为画布，准备用 D3.js 在上面作画：

复制代码

```
1 <svg></svg>
```

定义一个作画方法 `render`，任何时候我们希望针对改变的数据，重新更新画布，只需要调用下面定义的 `render` 方法：

复制代码

```
1 let render = (data) => {
2   // 选择节点
3   var circles = d3
4     .select('svg')
5     .selectAll('circle');
6
7   // 默认行为，对应于 update
8   circles.data(data)
9     .attr('r', 20)
10    .attr('cx', (d, i) => { return i * 50 + 20; })
11    .attr('cy', (d, i) => { return 20; })
12    .style('fill', 'BLUE')
13
14   // 新 data 加入，对应于 enter
15   circles.data(data)
16     .enter()
17     .append('circle')
18     .attr('r', 20)
19     .attr('cx', (d, i) => { return i * 50 + 20; })
```

```

20     .attr('cy', (d, i) => { return 20; })
21     .style('opacity', 0)
22     .style('fill', 'RED')
23     .transition()
24     .duration(1000)
25     .style('opacity', 1)
26
27 // 旧 data 离开，对应于 exit
28 circles.data(data)
29     .exit()
30     .transition()
31     .duration(1000)
32     .style('opacity', 0)
33     .remove();
34 };

```

你看，render 方法包含了这样几步：

首先，选择节点，即“萝卜坑”，在最开始的时候，一个坑也没有，即 svg 节点内没有任何 circle 节点；


第二步，定义了默认的 update 行为，在数据项，即萝卜保持占据萝卜坑的时候，进行的操作，在这里就是绘制蓝色的坑；

第三步，定义 enter 行为，即对于新来的萝卜，无法找到相应萝卜坑的时候，进行的操作，例子中就是建立新的红色的坑；

第四步，定义 exit 行为，当有萝卜要离开萝卜坑的时候，需要进行的操作，例子中就是删掉原有的坑。

其中链式调用中的 transition() 定义了在执行某些过程时，以过渡动画的方式来进行，例子中无论是“挖坑”还是“填坑”，都通过透明度渐变的方法来实现过渡。

最后，我们在第 0 秒的时候种下了 3 个萝卜，由于之前没有萝卜坑，于是发生了三次 enter 行为；第 2 秒的时候我们将萝卜减少到了 2 个，于是发生了一次 exit 行为；在第 4 秒的时候我们将萝卜数量变为 4 个，于是发生两次 enter 行为：

 复制代码

```

1 render([1, 2, 3]);
2 setTimeout(() => { render([1, 2]); }, 2000);
3 setTimeout(() => { render([1, 2, 3, 4]); }, 4000);

```

效果如下：



不知道你是否还记得我们在 [\[第 16 讲\]](#) 中介绍过的 Redux，D3.js 的这种机制和 Redux 的状态管理有着相似和相通之处。**状态都在统一的地方维护，而状态的改变，都通过事件的发生和响应机制来进行，且都将事件的响应逻辑（回调）交给用户来完成。**其实，这是一种很常见的“套路”，我们在后面的学习中，还将见到它的实现。

## 总结思考

今天，我们学习了 Web 绘图标准的基础知识，比较了 SVG 和 Canvas 这两种具备互补性的技术实现；同时，我们也学习了 Flot 和 D3.js 这两个差异很大，但都具备代表性的可视化 JavaScript 库。

希望你除了这两项同类技术之间孰优孰劣的比较以外，还掌握了不同类型技术之间联系比较的方法。随着学习的进行，对不同类型技术慢慢具备“深入”和“浅出”两个方向的理解，逐渐将充满关联的知识体系网状结构建立起来。

最后，我来提两个问题，供你思考一下吧：

思考一下你经历过的比较大的项目，你是否在项目中使用过数据可视化技术，如果给你一个机会，你觉得该怎样使用呢？

相信你用过 Google 地图或 Baidu 地图吧，那么，你觉得地图应用应该用 SVG 还是 Canvas 来实现呢，为什么？

## 扩展阅读

对于 SVG 和 Canvas 技术上的详细类比，我推荐你阅读 [SVG vs canvas: how to choose](#) 这篇文章。

学习数据可视化的技术有一个学习的小窍门，就是在掌握最基本的原理之后，可以直接跳到例子中去学习。作为可视化的库，对于其视觉上反馈迅速的特点，我们可以利用起来。比如文中提到的这两个库，Flot 提供了一些[实用的例子](#)，而 [D3.js 的例子](#)则是非常震撼。



# 全栈工程师修炼指南

从全栈入门到技能实战

熊燚

Oracle 首席软件工程师



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 17 | 不一样的体验：交互设计和页面布局

## 精选留言 (1)

写留言



Dream.

2019-10-21

#1思考一下你经历过的比较大的项目，你是否在项目中使用过数据可视化技术，如果给你一个机会，你觉得该怎样使用呢？

数据可视化，关键是将用户关注，并且值得分析的数据展示给用户。比如业务系统的业务量，运维系统的监控状态等等。...

展开 ▾





