

02 | 指令式绘图系统：如何用Canvas绘制层次关系图？

2020-06-24 月影

跟月影学可视化

[进入课程 >](#)



讲述：月影

时长 20:10 大小 18.47M



你好，我是月影。

Canvas 是可视化领域里非常重要且常用的图形系统，在可视化项目中，它能够帮助我们将数据内容以几何图形的形式，非常方便地呈现出来。

今天，我们就在上一节课的基础上对 Canvas 进行稍微深入一些的介绍，来学习一下 Canvas 绘制基本几何图形的方法。

我主要会讲解如何用它的 2D 上下文来完成绘图，不过，我不会去讲和它有关的所有，重点只是希望通过调用一些常用的 API 能给你讲清楚，Canvas2D 能做什么、要怎么使用，以及它的局限性是什么。最后，我还会带你用 Canvas 绘制一个表示省市关系的层次关



系图 (Hierarchy Graph)。希望通过这个可视化的例子，能帮你实践并掌握 Canvas 的用法。


在我们后面的课程中，基本上 70~80% 的图都可以用 Canvas 来绘制，所以其重要性不言而喻。话不多说，让我们正式开始今天的内容吧！

如何用 Canvas 绘制几何图形？

首先，我们通过一个绘制红色正方形的简单例子，来讲一讲 Canvas 的基本用法。

1. Canvas 元素和 2D 上下文

对浏览器来说，Canvas 也是 HTML 元素，我们可以用 canvas 标签将它插入到 HTML 内容中。比如，我们可以在 body 里插入一个宽、高分别为 512 的 canvas 元素。

 复制代码

```
1 <body>
2   <canvas width="512" height="512"></canvas>
3 </body>
```

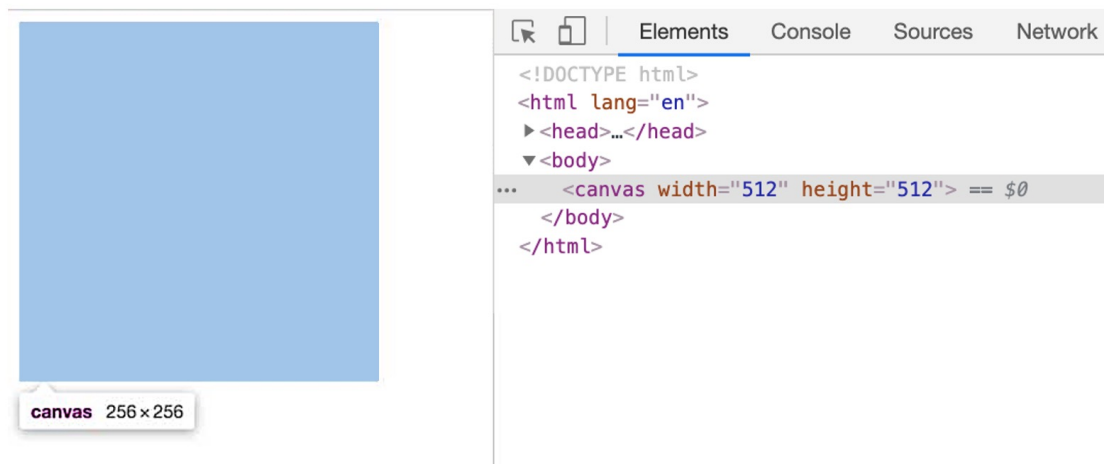
这里有一点需要特别注意，Canvas 元素上的 width 和 height 属性不等同于 Canvas 元素的 CSS 样式的属性。这是因为，CSS 属性中的宽高影响 Canvas 在页面上呈现的大小，而 HTML 属性中的宽高则决定了 Canvas 的坐标系。为了区分它们，我们称 Canvas 的 HTML 属性宽高为**画布宽高**，CSS 样式宽高为**样式宽高**。

在实际绘制的时候，如果我们不设置 Canvas 元素的样式，那么 Canvas 元素的画布宽高就会等于它的样式宽高的像素值，也就是 512px。

而如果这个时候，我们通过 CSS 设置其他的值指定了它的样式宽高。比如说，我们将样式宽高设置成 256px，那么它实际的画布宽高就是样式宽高的两倍了。代码和效果如下：

 复制代码

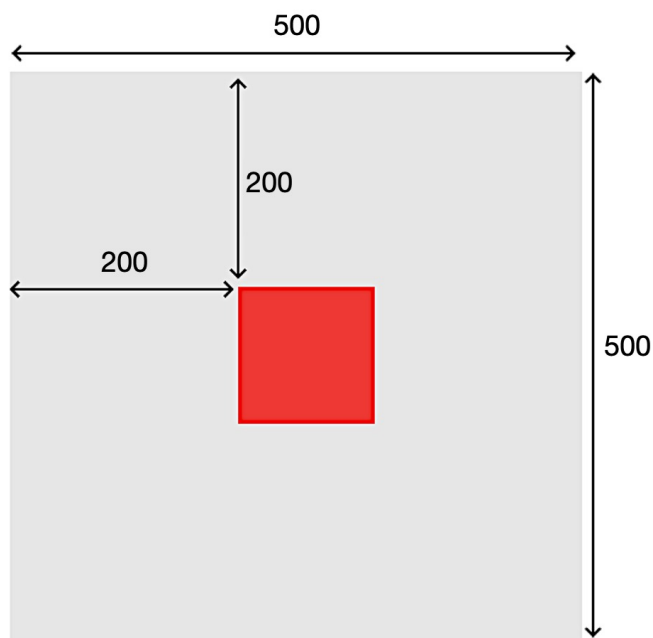
```
1 canvas {
2   width: 256px;
3   height: 256px;
4 }
```



将canvas的样式宽高设为256px，画布宽高设为512px

因为画布宽高决定了可视区域的坐标范围，所以 Canvas 将画布宽高和样式宽高分开的做法，能更方便地适配不同的显示设备。

比如，我们要在画布宽高为 500*500 的 Canvas 画布上，绘制一个居中显示的 100*100 宽高的正方形。我们只要将它的坐标设置在 $x = 200, y = 200$ 处即可。这样，不论这个 Canvas 以多大的尺寸显示在各种设备上，我们的代码都不需要修改。否则，如果 Canvas 的坐标范围（画布宽高）跟着样式宽高变化，那么当屏幕尺寸改变的时候，我们就要重新计算需要绘制的图形的所有坐标，这对于我们来说将会是一场“灾难”。

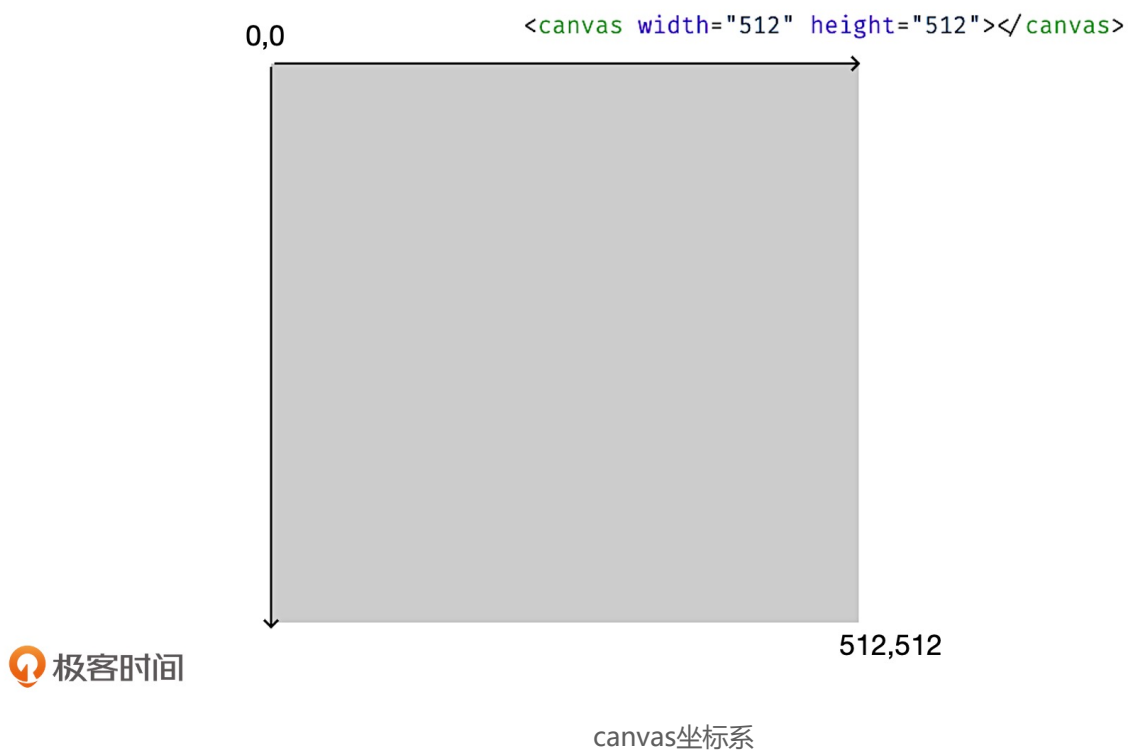


在画布宽高为500X500的Canvas上居中绘制一个100*100的正方形

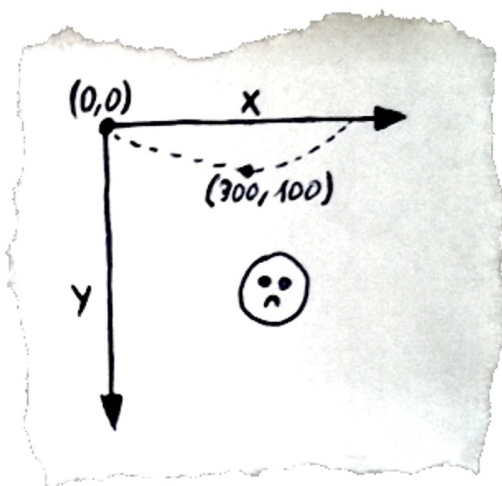
2. Canvas 的坐标系

好了，Canvas 画布已经设置好了，接下来我们来了解一下 Canvas 的坐标系。

Canvas 的坐标系和浏览器窗口的坐标系类似，它们都默认左上角为坐标原点，x 轴水平向右，y 轴垂直向下。那在我们设置好的画布宽高为 512*512 的 Canvas 画布中，它的左上角坐标值为 (0,0)，右下角坐标值为 (512,512)。这意味着，坐标 (0,0) 到 (512,512) 之间的所有图形，都会被浏览器渲染到画布上。

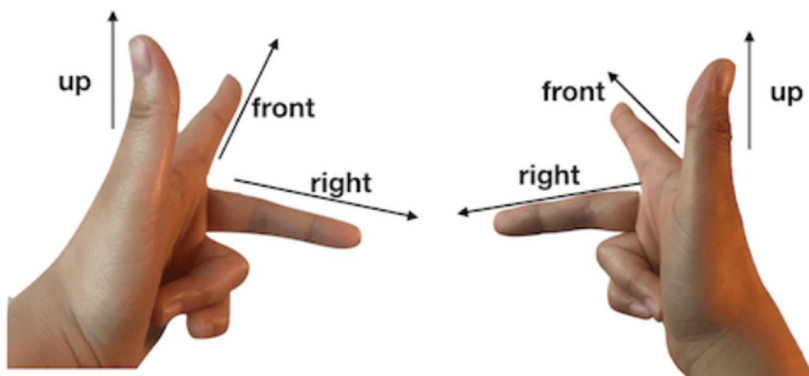


注意，上图中这个坐标系的 y 轴向下，意味着这个坐标系和笛卡尔坐标系不同，它们的 y 轴是相反的。那在实际应用的时候，如果我们想绘制一个向右上平抛小球的动画，它的抛物线轨迹，在 Canvas 上绘制出来的方向就是向下凹的。



Canvas坐标系

另外，如果我们再考虑旋转或者三维运动，这个坐标系就会变成“左手系”。而左手系的平面法向量的方向和旋转方向，和我们熟悉的右手系相反。如果你现在还不能完全理解它们的区别，那也没关系，在实际应用的时候，我会再讲的，这里你只需要有一个大体印象就可以了。



左手系和右手系


3. 利用 Canvas 绘制几何图形

有了坐标系，我们就可以将几何图形绘制到 Canvas 上了。具体的步骤可以分为两步，分别是获取 Canvas 上下文和利用 Canvas 上下文绘制图形。下面，我们——来看。

第一步，获取 Canvas 上下文。

那在 JavaScript 中，我们要获取 Canvas 上下文也需要两个步骤。首先是获取 Canvas 元素。因为 Canvas 元素就是 HTML 文档中的 canvas 标签，所以，我们可以通过 DOM API 获取它，代码如下：

```
1 const canvas = document.querySelector('canvas');
```

 复制代码

获取了 canvas 元素后，我们就可以通过 getContext 方法拿到它的上下文对象。具体的操作就是，我们调用 canvas.getContext 传入参数 2d。

 复制代码

```
1 const context = canvas.getContext('2d');
```

有了 2d 上下文，我们就可以开始绘制图形了。

第二步，用 Canvas 上下文绘制图形。

我们拿到的 context 对象上会有许多 API，它们大体上可以分为两类：一类是设置状态的 API，可以设置或改变当前的绘图状态，比如，改变要绘制图形的颜色、线宽、坐标变换等等；另一类是绘制指令 API，用来绘制不同形状的几何图形。

如何使用这些 API 呢？我来举个例子，假设我们要在画布的中心位置绘制一个 100*100 的红色正方形。那我们该怎么做呢？

首先，我们要通过计算得到 Canvas 画布的中心点。前面我们已经说过，Canvas 坐标系的左上角坐标是 (0,0)，右下角是 Canvas 的画布坐标，即

(canvas.width,canvas.height)，所以画布的中心点坐标是 (0.5 * canvas.width, 0.5 * canvas.height)。

如果我们要在中心点绘制一个 100 * 100 的正方形，那对应的 Canvas 指令是：

 复制代码

```
1 context.rect(0.5 * canvas.width, 0.5 * canvas.height, 100, 100);
```

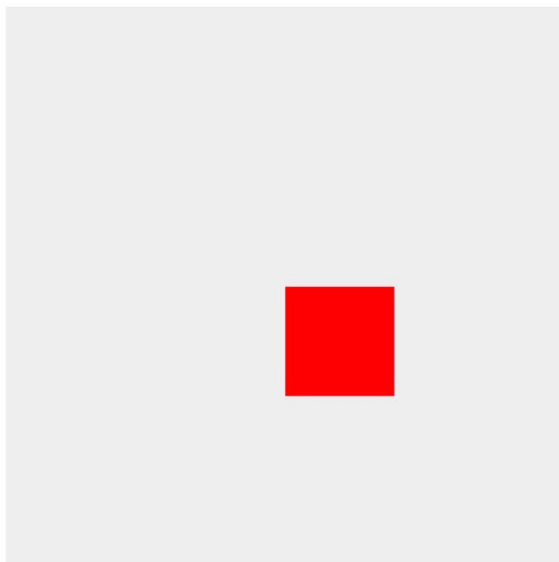
其中，context.rect 是绘制矩形的 Canvas 指令，它的四个参数分别表示要绘制的矩形的 x、y 坐标和宽高。在这里我们要绘制的正方形宽高都是 100，所以后两个参数是 100 和 100。

那在实际绘制之前，我们还有一些工作要做。我要将正方形填充成红色，这一步通过调用 context.fillStyle 指令就可以完成。然后，我们要调用一个 beginPath 的指令，告诉

Canvas 我们现在绘制的路径。接着，才是调用 rect 指令完成绘图。最后，我们还要调用 fill 指令，将绘制的内容真正输出到画布中。这样我们就完整了绘制，绘制的效果和代码如下：

 复制代码

```
1 const rectSize = [100, 100];
2 context.fillStyle = 'red';
3 context.beginPath();
4 context.rect(0.5 * canvas.width, 0.5 * canvas.height, ...rectSize);
5 context.fill();
```



但是，看到上面这张图，我们发现了一个问题：正方形并没有居于画布的正中心。这是为什么呢？

你可以回想一下我们刚才的操作，在绘制正方形的时候，我们将 rect 指令的参数 x、y 设为画布宽高的一半。而 rect 指令的 x、y 的值表示的是，我们要绘制出的矩形的左上角坐标而不是中心点坐标，所以绘制出来的正方形自然就不在正中心了。

那我们该如何将正方形的中心点放在画布的中心呢？这就需要我们移动一下图形中心的坐标了。具体的实现方法有两种。

第一种做法是，我们可以让 rect 指令的 x、y 参数，等于画布宽高的一半分别减去矩形自身宽高的一半。这样，我们就把正方形的中心点真正地移动到画布中心了。代码如下所示：

 复制代码

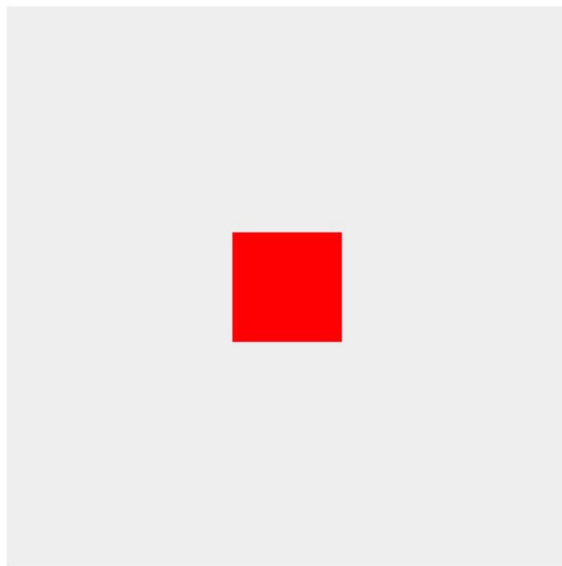
```
1 context.rect(0.5 * (canvas.width - rectSize[0]), 0.5 * (canvas.height - rectSi:
```

第二种做法是，我们可以先给画布设置一个平移变换（Translate），然后再进行绘制。代码如下所示：

 复制代码

```
1 context.translate(-0.5 * rectSize[0], -0.5 * rectSize[1]);  
2
```

在上面的代码中，我们给画布设置了一个平移，平移距离为矩形宽高的一半，这样它的中心点就是画布的中心了。




既然这两种方法都能将图形绘制到画布的中心，那我们该怎么选择呢？其实，我们可以从这两种方式各自的优缺点入手，下面我就具体来说一说。

第一种方式很简单，它直接改变了我们要绘制的图形顶点的坐标位置，但如果我们绘制的不是矩形，而是很多顶点的多边形，我们就需要在绘图前重新计算出每个顶点的位置，这会非常麻烦。

第二种方式是对 Canvas 画布的整体做一个平移操作，这样我们只需要获取中心点与左上角的偏移，然后对画布设置 `translate` 变换就可以了，不需要再去改变图形的顶点位置。不过，这样一来我们就改变了画布的状态。如果后续还有其他的图形需要绘制，我们一定要记得把画布状态给恢复回来。好在，这也不会影响到我们已经画好的图形。

那怎么把画布状态恢复回来呢？恢复画布状态的方式有两种，**第一种是反向平移**。反向平移的原理也很简单，你可以直接看下面的代码。

 复制代码

```
1 // 平移
2 context.translate(-0.5 * rectSize[0], -0.5 * rectSize[1]);
3
4 ... 执行绘制
5
6 // 恢复
7 context.translate(0.5 * rectSize[0], 0.5 * rectSize[1]);
```

除了使用反向平移的恢复方式以外，Canvas 上下文还提供了 **save** 和 **restore** 方法，可以暂存和恢复画布某个时刻的状态。其中，`save` 指令不仅可以保存当前的 `translate` 状态，还可以保存其他的信息，比如，`fillStyle` 等颜色信息。而 `restore` 指令则可以将状态指令恢复成 `save` 指令前的设置。操作代码如下：

 复制代码

```
1 context.save(); // 暂存状态
2 // 平移
3 context.translate(-0.5 * rectSize[0], -0.5 * rectSize[1]);
4
5 ... 执行绘制
6
7 context.restore(); // 恢复状态
```

好了，把一个简单矩形绘制到画布中心的完整方法，我们已经说完了。那我们来回顾一下利用 Canvas 绘制矩形的过程。我把这个过程总结为了 5 个步骤：

1. 获取 Canvas 对象，通过 `getContext('2d')` 得到 2D 上下文；
2. 设置绘图状态，比如填充颜色 `fillStyle`，平移变换 `translate` 等等；
3. 调用 `beginPath` 指令开始绘制图形；
4. 调用绘图指令，比如 `rect`，表示绘制矩形；
5. 调用 `fill` 指令，将绘制内容真正输出到画布上。

除此之外，Canvas 上下文还提供了更多丰富的 API，可以用来绘制直线、多边形、弧线、圆、椭圆、扇形和贝塞尔曲线等等，这里我们不一一介绍了。在之后的课程中，我们会详细讲解如何利用这些 API 来绘制复杂的几何图形。如果你还想了解更多关于 Canvas 的 API 相关的知识，还可以去阅读 [MDN 文档](#)。

如何用 Canvas 绘制层次关系图？

知道了 Canvas 的基本用法之后，接下来，我们就可以利用 Canvas 给一组城市数据绘制一个层次关系图了。也就是在一组给出的层次结构数据中，体现出同属于一个省的城市。

在操作之前呢，我们先引入一个概念层次结构数据（Hierarchy Data），它是可视化领域的专业术语，用来表示能够体现层次结构的信息，例如城市与省与国家。一般来说，层次结构数据用层次关系图表来呈现。

其中，城市数据是一组 JSON 格式的数据，如下所示。

 复制代码

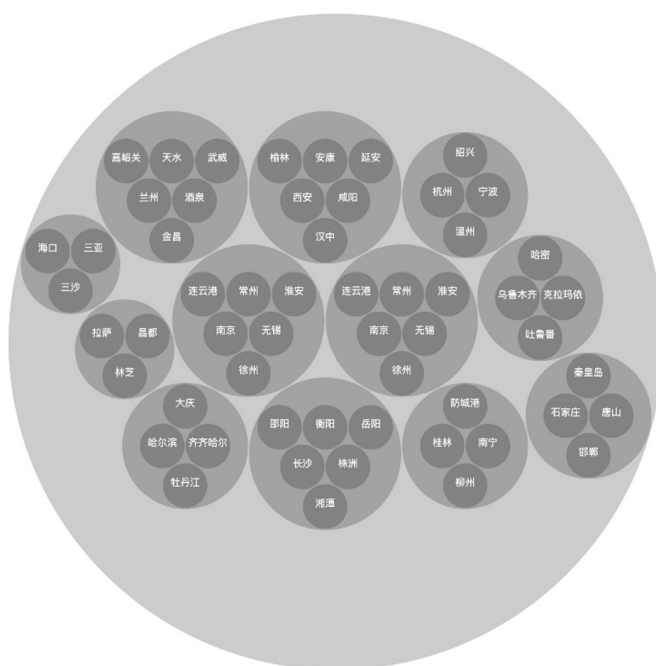
```
1 {  
2   "name": "中国",  
3   "children":  
4     [  
5       {  
6         "name": "浙江",  
7         "children":  
8           [  
9             { "name": "杭州" },  
10            { "name": "宁波" },  
11            { "name": "温州" },  
12            { "name": "绍兴" }  
13          ]  
14        },  
15        {  
16          "name": "广西",
```

```


17     "children":
18     [
19         {"name": "桂林"},
20         {"name": "南宁"},
21         ...
22     ]
23 ]
24 }

```

我们要绘制的层次关系图效果如下：



知道了要求之后，我们应该怎么做呢？首先，我们要从数据源获取 JSON 数据。

 复制代码

```

1  const dataSource = 'https://s5.ssl.qhres.com/static/b0695e2dd30daa64.json';
2  (async function () {
3      const data = await (await fetch(dataSource)).json();
4
5  })();

```

这份 JSON 数据中只有“城市 > 省份 > 中国”这样的层级数据，我们要将它与绘图指令建立联系。建立联系指的是，我们要把数据的层级、位置和要绘制的圆的半径、位置——对应起来。

换句话说，我们要把数据转换成图形信息，这个步骤需要数学计算。不过，我们可以直接使用 d3-hierarchy [d3-hierarchy](#) 这个工具库转换数据。


 复制代码

```
1 (async function () {
2   const data = await (await fetch(dataSource)).json();
3
4   const regions = d3.hierarchy(data)
5     .sum(d => 1)
6     .sort((a, b) => b.value - a.value);
7
8   const pack = d3.pack()
9     .size([1600, 1600])
10    .padding(3);
11
12   const root = pack(regions);
13 }());
```

数学计算的内容，我会在数据篇中详细来讲。这里，我们就先不介绍 d3 的具体转换实现了，你只需要知道，我们可以用 `d3.hierarchy(data).sum(...).sort(...)` 将省份数据按照包含城市的数量，从多到少排序就可以了。

假设，我们要将它们展现在一个画布宽高为 1600 * 1600 的 Canvas 中，那我们可以通过 `d3.pack()` 将数据映射为一组 1600 宽高范围内的圆形。不过，为了展示得美观一些，在每个相邻的圆之间我们还保留 3 个像素的 padding（按照经验，我们一般是保留 3 个像素 padding 的，但这也要根据实际的设计需求来更改）。

这样，我们就获得了包含几何图形信息的数据对象。此时它的内部结构如下所示：

 复制代码

```
1 {
2   data: {name: '中国', children: [...]},
3   children: [
4     {
5       data: {name: '江苏', children: [...]},
6       value: 7,
7       r: 186.00172579386546,
8       x: 586.5048250548921,
9       y: 748.2441892254667,
10    }
11    ...
12  ],
```

```
13   value: 69,  
14   x: 800,  
15   y: 800,  
16   r: 800,  
17 }
```

我们需要的信息是数据中的 x 、 y 、 r ，这些数值是前面调用 `d3.hierarchy` 帮我们算出来的。接下来我们只需要用 Canvas 将它们绘制出来就可以了。具体绘制过程比较简单，**只需要遍历数据并且根据数据内容绘制圆弧**，我也把它总结成了两步。

第一步：我们在当前数据节点绘制一个圆，圆可以使用 `arc` 指令来绘制。`arc` 方法的五个参数分别是圆心的 x 、 y 坐标、半径 r 、起始角度和结束角度，前三个参数就是数据中的 x 、 y 和 r 。因为我们要绘制的是整圆，所以后面的两个参数中起始角是 0 ，结束角是 2π 。

第二步，绘制图成后，如果这个数据节点有下一级数据，我们遍历它的下一级数据，然后递归地对这些数据调用绘图过程。如果没有下一级数据，说明当前数据为城市数据，那么我们就直接给出城市的名字，这一步可以通过 `fillText` 指令来完成。具体的代码如下所示：

 复制代码

```
1  const TAU = 2 * Math.PI;  
2  
3  function draw(ctx, node, {fillStyle = 'rgba(0, 0, 0, 0.2)', textColor = 'white'  
4    const children = node.children;  
5    const {x, y, r} = node;  
6    ctx.fillStyle = fillStyle;  
7    ctx.beginPath();  
8    ctx.arc(x, y, r, 0, TAU);  
9    ctx.fill();  
10   if(children) {  
11     for(let i = 0; i < children.length; i++) {  
12       draw(context, children[i]);  
13     }  
14   } else {  
15     const name = node.data.name;  
16     ctx.fillStyle = textColor;  
17     ctx.font = '1.5rem Arial';  
18     ctx.textAlign = 'center';  
19     ctx.fillText(name, x, y);  
20   }  
21 }  
22  
23 draw(context, root);
```

这样，我们就用 Canvas 绘图简单地实现了一个层次关系图，它的代码不多也不复杂，你可以很容易理解，所以我就不做过多的解释啦。

Canvas 有哪些优缺点？

通过上面的例子，相信你已经熟悉了 Canvas 的基础用法。但是，浏览器是一个复杂的图形环境，要想灵活使用 Canvas，我们还需要从宏观层面来知道，它能做什么，不能做什么。

简单来说，就是要了解 Canvas 的优缺点。

首先，Canvas 是一个非常简单易用的图形系统。结合刚才的例子你也能感受到，Canvas 通过一组简单的绘图指令，就能够方便快捷地绘制出各种复杂的几何图形。

另外，Canvas 渲染起来相当高效。即使是绘制大量轮廓非常复杂的几何图形，Canvas 也只需要调用一组简单的绘图指令就能高性能地完成渲染。这个呀，其实和 Canvas 更偏向于渲染层，能够提供底层的图形渲染 API 有关。那在实际实现可视化业务的时候，Canvas 出色的渲染能力正是它的优势所在。

不过 Canvas 也有缺点，因为 Canvas 在 HTML 层面上是一个独立的画布元素，所以所有的绘制内容都是在内部通过绘图指令来完成的，绘制出的图形对于浏览器来说，只是 Canvas 中的一个像素点，我们很难直接抽取其中的图形对象进行操作。

比如说，在 HTML 或 SVG 中绘制一系列图形的时候，我们可以一一获取这些图形的元素对象，然后给它们绑定用户事件。但同样的操作在 Canvas 中没有可以实现的简单方法（但是我们仍然可以和 Canvas 图形交互，在后续课程中我们会有专门讨论）。下一节课中，我们会详细讲解 SVG 图形系统，到时你就会更加明白它们的差异具体是什么了。

要点总结

好了，Canvas 的使用讲完了，我们来总结一下你要掌握的重点内容。

首先，我们讲了利用 Canvas 绘制几何图形，这个过程很简单，不过依然有 3 点需要我们注意：

1. 在 HTML 中建立画布时，我们要分别设置画布宽高和样式宽高；

2. 在建立坐标系时，我们要注意 canvas 的坐标系和笛卡尔坐标系在 y 轴上是相反的；
3. 如果要把图形绘制在画布中心，我们不能直接让 x、y 的坐标等于画布中心坐标，而是要让图形中心和画布中心的位置重叠。这个操作，我们可以通过计算顶点坐标或者 平移变换来实现。

接着，我们讲了利用 Canvas 展示数据的层级关系。在这个过程中，我们应当先处理数据，将数据内容与绘图指令建立映射关系，然后遍历数据，通过映射关系将代表数据内容的参数传给绘图指令，最后将图形绘制到 Canvas 上。

另外，我们还讲了 Canvas 的优缺点。在实际实现可视化业务的时候，Canvas 的简单易操作和高效的渲染能力是它的优势，但是它的缺点是不能方便地控制它内部的图形元素。

最后，我还有一点想要补充一下。我们今天绘制的图形都是静态的，如果要使用 Canvas 绘制动态的图形也很简单，我们可以通过 clearRect 指令，将之前的图形擦除，再把新的图形绘制上去即可。在后续课程中，我们有专门的章节来介绍动画。

小试牛刀

最后呢，我为你准备了两道课后题，试着动手操作一下吧！

1. 这节课我们介绍了用 Canvas 绘制矩形和圆弧，实际上 Canvas 还有更多的绘图指令来绘制不同类型的几何图形。你可以试着修改一下前面显示正方形的例子，在画布的中心位置显示一个三角形、椭圆或五角星。
2. Canvas 的缺点是不能方便地控制它内部的图形元素，但这不代表它完全不能控制。你可以尝试给我们前面绘制的层次关系图增加鼠标的交互，让鼠标指针在移动到某个城市所属的圆的时候，这个圆显示不同的颜色（提示：你可以获取鼠标坐标，然后用这个坐标到圆心的距离来判断）。

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！

源码

[🔗 用 Canvas 绘制层次关系图的完整代码](#)

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 01 | 浏览器中实现可视化的四种方式

精选留言

 写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。