Canvas 画布

陈一帅

实务学堂

介绍

- HTML5 canvas API
- 画布是绘图用的单个DOM元素
- 它提供一个编程接口，用于在节点占用的空间上绘制图像
- 使用JavaScript绘制光栅图像
- 可用于动画、游戏、数据可视化、图片编辑以及实时视频处理

绘图样式

- 两种广泛支持的绘图样式
- Canvas API 绘制二维图形
- WebGL API 绘制三维图形
  - 通过OpenGL接口绘制三维图形
  - 硬件加速
  - 允许用JavaScript有效渲染复杂场景

和SVG区别

- SVG中，保留了形状的原始描述，以便可以移动、放大
- Canvas在画布绘制后，立即将其转换为像素（光栅上的彩色点）
  - 不记得这些像素代表什么
  - 移动图像的唯一方法是清除画布（或画布中围绕该形状的部分）并在新位置重新绘制形状

canvas元素

- 图形绘制到 canvas 元素上
- 一个HTML页面可以包含多个画布元素
- 一个新的画布是空的，完全透明，在文档中显示为空白
  - 没有边框，没有内容

绘制

- 所有绘制都用JavaScript完成
- 首先通过DOM查询引用画布
- 然后使用其getContext（）得到JavaScript上下文对象（context），它能动态创建图像
  - 画布是笛卡尔网格，其左上角的坐标为（0，0）

```
const canvas = document.getElementById('drawing');
const context = canvas.getContext('2d');
context.moveTo(0, 0);
context.lineTo(canvas.width, canvas.height);
context.stroke();
```

示例

方法

- 绘制线条，圆弧，文本，颜色和图像
- 基本形状
  - 画布仅支持两种基本形状：矩形和椭圆形
- 路径
  - 所有其他形状必须通过组合一个或多个路径来创建
  - canvas API中有多种路径方法，可以用来组合形状

线条 Line

- lineTo(x, y)
- 绘制一条从当前位置到指定x以及y位置的直线。

```
context.lineTo(canvas.width, canvas.height);
context.stroke(); // 画线
```

示例

弧线

- arc(x, y, radius, startAngle, endAngle, anticlockwise)
  - 画一个以（x,y）为圆心的以 radius 为半径的圆弧
  - 从 startAngle （开始角度）到 endAngle （结束角度）
  - 按照 anticlockwise （时钟方向）给定的方向（默认为顺时针）来生成

```
context.arc(canvas.width/2,
            canvas.height/2, 125, 0, 4);
```

贝塞尔曲线

- quadraticCurveTo(cp1x, cp1y, x, y)
  - 二次贝塞尔曲线
  - cp1x,cp1y为一个控制点
  - x,y为结束点
- bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)
  - 三次贝塞尔曲线
  - cp1x,cp1y为控制点一
  - cp2x,cp2y为控制点二
  - x,y为结束点

形状

- 画布仅支持两种基本形状
  - 矩形和椭圆形
- 可以填充（fill）形状
  - 给填充区域指定某种颜色或图案
- 也可以描边（stroke）
  - 沿着边缘绘制一条线

矩形 Rect

- 三种方法
  - fillRect(x, y, width, height) 绘制一个填充的矩形
  - strokeRect(x, y, width, height) 绘制一个矩形的边框
  - clearRect(x, y, width, height) 清除指定矩形区域，让清除部分完全透明
- 参数
  - x与y指定制矩形的左上角（相对于原点）坐标
  - width和height设置矩形尺寸

矩形 Rect

- 说明
  - fillRect() 绘制一个填充的矩形
  - clearRect() 从正方形中心开始擦除了一个正方形
  - strokeRect() 生成一个正方形边框
- 绘制之后会马上显现在canvas上，即时生效

```
context.fillRect(100, 25, 200, 200);
context.clearRect(125, 50, 200, 200);
context.strokeStyle = 'rgb(255, 200, 0)';
context.strokeRect(150, 75, 200, 200);
```

示例

椭圆

- ellipse(x, y, radiusX, radiusY, rotation, startAngle, endAngle, anticlockwise);
  - 椭圆圆心在（x,y）位置
  - 半径分别是radiusX 和 radiusY
  - 从 startAngle 开始绘制，到 endAngle 结束
  - 按照anticlockwise（默认顺时针）指定的方向

```
context.ellipse(canvas.width/2,
        canvas.height/2, 100, 100, 0, 0,
        2 * Math.PI, false);
```

Path

- 图形的基本元素是路径
  - 路径是通过不同颜色和宽度的线段或曲线相连形成的不同形状的点的集合
  - 一个路径，甚至一个子路径，都是闭合的
- 步骤
  - 使用路径绘制图形需要一些额外的步骤
  - 首先创建路径起始点，然后画出路径，之后把路径封闭
  - 然后描边或填充路径区域来渲染图形

SVG Path

- Path2D

```
let path = new Path2D("
    M 20 40
    L 100 20
    L 175 125
    L 120 180 z");
```

示例

Path

- beginPath()
  - 新建一条路径，图形绘制命令被指向到路径上生成路径
- closePath()
  - 闭合路径，绘制一条从当前点到开始点的直线来闭合图形
  - 之后图形绘制命令又重新指向到上下文中

```
context.beginPath();
context.moveTo(225, 50);
context.lineTo(350, 250);
context.closePath();
```

示例

Path

- stroke()
  - 描边
- fill()
  - 填充

```
context.fillStyle = gradient;
context.beginPath();
context.moveTo(225, 50);
context.lineTo(350, 250);
context.closePath();
context.stroke();
context.fill();
```

示例

Path fill 填充

- 必须先关闭路径，使起点和终点就在同一位置，然后才能填充它
- 如果尚未关闭路径，则从其末端到其起点添加一条线，并填充完成路径所包围的形状

```
context.fillStyle = gradient;
context.beginPath();
context.moveTo(225, 50);
context.lineTo(350, 250);
context.closePath();
context.stroke();
context.fill();
```

示例

循环绘画

- 利用编程的强大能力，自动绘画

示例

大小设置

- Canvas 画布是网页上的一个矩形区域
- 默认大小为300×150像素（宽×高）
  - 可以设定 canvas 元素的width和height属性，以确定其大小（以像素为单位）

```
canvas.style.height = height + 'px';
let scale = window.devicePixelRatio;
canvas.width = width * scale;
```

示例

放大和缩小

- window.devicePixelRatio 存着网页的放大倍数
  - 网页放大缩小时，会变化
- 设置 context.scale，这样图像大小也会随着网页的放大/缩小而变化

```
canvas.style.width = referenceWidth + 'px';

let scale = window.devicePixelRatio;
canvas.width = referenceWidth * scale;

context.scale(scale, scale);
```

示例

绘制图片

- drawImage，按照一副图像或其他画布，绘制我们的画布
  - 像素级复制
- 默认绘制整个源图像
  - 提供更多参数，可以复制图像的特定区域
- 应用：游戏编程
  - 从包含各种姿势的图像中复制游戏角色的各种图像，来获得游戏的动态效果

```
context.drawImage(image, 0, 0, 450, 300);
```

示例

图像处理

- 图像中的每个元素，以R，G，B，顺序存在一个数组中
- 设置随机像素

示例

图像处理

- 逐渐增加绿色和蓝色

```
for (let y = 0; y < canvas.height; y++) {
  for (let x = 0; x < canvas.width; x++) {
    let index = (x + y * canvas.width) * 4;
    pixelData[index + 0] = 255; // red
    pixelData[index + 1]
      = (x / canvas.width) * 255; // green
    pixelData[index + 2]
      = (y / canvas.height) * 255; // blue
    pixelData[index + 3] = 255; // alpha
    }
}
```

示例

图像处理

- 处理每一个像素
- 颜色反转

```
let imageData = context.getImageData
        (0, 0, canvas.width, canvas.height);
let data = imageData.data;
for (let i = 0; i < data.length; i += 4) {
    data[i] = 255 - data[i]; // red
    data[i + 1] = 255 - data[i + 1]; // green
    data[i + 2] = 255 - data[i + 2]; // blue
}
context.putImageData(imageData, 0, 0);
```

示例

MDN

文字

- 指定文字字体等性质

```
context.font = '44px Helvetica';
context.textBaseline = 'middle';
context.textAlign = 'center';
context.strokeStyle = 'black';
context.strokeText('科蚪加油! ', width/2, height/2);
```

示例

文字

- 窗口大小变化时，重画
  - 重新取窗口大小

```
window.addEventListener('resize',
    function() {
        setup();
        draw();
});
```

示例

状态切换

- 平移，缩放和旋转，会改变 context 状态
  - 影响所有后续绘图操作
- 所以，在做它之前，可以使用save方法保存变换前的状态
  - 完成之后，再使用restore方法恢复变换前的状态

```
context.save(); // 保存当前坐标系
context.translate(50, 0); // 向右移动（对于SVG路径）
context.fill(path); // 画出Path
context.restore(); // 恢复坐标系
```

示例2

遮盖方式

- 之前例子里，总是将一个图形画在另一个上
- 可利用 globalCompositeOperation 属性来改变
  - 设定遮盖策略，12种遮盖方式
- clip裁切
  - 将当前正在构建的路径转换为当前的裁剪路径
  - 隐藏不想看到的部分图形

```
context.globalCompositeOperation = 'difference';
```

示例

MDN

动画

- 定时重绘，得到动画效果
- 两种定时的方法
  - setInterval 和 setTimeout 定时重绘
  - requestAnimationFrame（）方法非常适合画布动画

绘制步骤

- 先清除画布
  - clearRect 清空 canvas 所有内容
- 保存画布状态
  - 如果要改变一些会改变 canvas 状态的设置（样式，变形之类的），又要在每画一帧之时都是原始状态的话，你需要先保存一下
- 绘制动画形状
- 恢复画布状态，重绘下一帧

示例

示例1

示例2

MDN

限制

- 图像一旦绘制出来，就一直保持那样了
- 如果需要移动它，不得不对所有东西（包括之前的）进行重绘
- 重绘相当费时，性能很依赖于电脑的速度

视频处理

- play播放视频

```
video.play();

function draw() {
    context.drawImage(video, 0, 0, 533, 300);
    requestAnimationFrame(draw);
}

video.addEventListener('play', draw);
```

示例1

视频处理（特效）

- 类似图像处理，也可以对视频图像的逐帧进行处理

```
// video image processing
for (let i = 0; i < data.length; i += 4) {
  data[i] = data[i + (canvas.width * 4 / 3)];
  data[i + 1]
    = data[i + 1 + (canvas.width * 4 / 1.5)];
  data[i + 2]
    = data[i + 2 + (canvas.width * 4)];
}
```

示例1

示例2

MDN

视频和SVG叠加

```
// SVG path data for each video pixel
let path = new Path2D("M 20 40 L 100 20 L 175 125 L 120 1
// 柔光复合
context.globalCompositeOperation = 'soft-light';
context.fill(path);
```

示例

注意

- 绘制到画布上的像素数据不是DOM元素
- 画布区域不能以与SVG图像区域相同的方式进行交互

参考

MDN Canvas API

MDN 入门指南

练习1

- MDN 上的动画
  - 太阳系的动画
  - 动画时钟
  - 循环全景照片
  - Snake Game

练习2

- MDN上的高级动画
  - 弹球

练习3

- 用这个模板创建一个HTML画布图形
- 每个画布为600×400（CSS）像素
- 使用窗口对象的devicePixelRatio属性来缩放
- 应包括使用canvas API的path方法绘制的自定义形状
- 应由ImageData对象处理的图像或视频
- 每个画布绘图都应使用窗口对象的requestAnimationFrame（）方法生成动画
- 用JavaScript更新页面标题和画布元素的背景色