绘图

管理科学与工程学科
耿方方

主要内容

- Canvas基础知识
- 图形绘制
- 图形变换与控制

应用领域

- 1、游戏
- 2、可视化数据：：百度的echart、d3.js、three.js
- 3、banner广告
- 4、未来

- 模拟器：无论从视觉效果还是核心功能方面来说，模拟器产品可以完全由JavaScript来实现。
- 远程计算机控制：Canvas可以让开发者更好地实现基于Web的数据传输，构建一个完美的可视化控制界面。
- 图形编辑器：Photoshop图形编辑器将能够100%基于Web实现。

1、Canvas基础知识

- 基本原理
- Canvas元素在页面上提供一块像画布一样无色透明的区域，可通过Javascript脚本绘制图形。
- 在HTML页面上定义Canvas元素除了可以指定id、style、class、hidden等通用属性之外，还可以指定以下2个属性：
- height：设置画布组件的高度。
- width：设置画布组件的宽度。

1、Canvas基础知识

- 在画布上绘制图形必须经过以下三个步骤：
- 获取Canvas对应的DOM对象，得到一个Canvas对象。
- 调用Canvas对象的getContext()方法，得到CanvasRenderingContext2D对象（可绘制图形）。
- 调用CanvasRenderingContext2D对象方法绘图。

1、Canvas基础知识

- Canvas API
- Canvas API通过调用Canvas对象的getContext()方法获得图形对象。
- 调用传入参数【2d】，返回的CanvasRenderingContext2D对象就是Canvas API对象实例，叫做2D渲染上下文。

1、Canvas基础知识

- 常见的绘图方法如下：

表 18-01 CanvasRenderingContext2D 绘图方法

方法	简要说明
void arc(flaot x, float y, float radius, float startAngel, endAngle, boolen counterclockwise)	向 Canvas 的当前路径上添加一段弧
void arcTo(float x1, float y1, float x2, float y2, float radius)	向 Canvas 的当前路径上添加一段弧，与前一个方法相比，只是定义弧的方式不同
void beginPath()	开始定义路径
void closePath()	关闭前面定义的路径
void bezierCurveTo(float cpX1, float cpY1, float cpX2, float cp Y2, float x, float y)	向 Canvas 的当前路径上添加一段贝塞尔曲线
void clearRect(float x, float y, float width, float height)	擦除制定区域上绘制的图形
void clip()	从画布上裁切一块出来
Canvas Gradient createLinearGradient(float xStart, float yStart, float xEnd, float yEnd)	创建一个线性渐变
CanvasPatterm careatePattern(image image, string style)	创建一个图形平铺
Canvas Gradient createLinearGradient(float xStart, float yStart, float radiusStart, float xEnd, float yEnd, float radiusEnd)	创建一个圆形渐变
void drawImage(Image image, float x, float y) void drawImage(Image image, float x, float y, float width, float height) void drawImage(Image image, integer sx, integer xy, integer sw, integer sh, float dx, float dy, float dw, float dh)	绘制位图

1、Canvas基础知识

- 常见的绘图方法如下：

表 18-01 CanvasRenderingContext2D 绘图方法

方法	简要说明
void fill()	填充 Canvas 的当前路径
void fillRect(float x, float y, float width, float height)	填充一个矩形区域
void fillText(String text, float x, float y [, float maxWidth])	填充字符串
void lineTo(float x, float y)	把 Canvas 的当前路径从当前结束点连接到 x、y 的对应点
void moveTo(float x, float y)	把 Canvas 的当前路径结束点移动到 x、y 对应的点
void quadraticCurveTo(float cpX, float cpY, float x, float y)	向 Canvas 当前路径上添加一段二次曲线
void rect(float x, float y, float width, float height)	向 Canvas 当前路径上添加一个矩形
void stroke()	沿着 Canvas 当前路径绘制边框
void strokeRect(float x, float y, float width, float height)	绘制一个矩形边框
void strokeText(string text, float x, float y, float width [,float maxWidth])	绘制字符串边框
void save()	保存当前绘图状态
void restore()	恢复之前保存的绘图状态
void rotate(float angle)	旋转坐标系统
void scale(float sx, float sy)	缩放坐标系统
void translate(float dy, float dy)	平移坐标系统

1、Canvas基础知识

- CanvasRenderingContext2D属性功能用法如下：

表 18-02 CanvasRenderingContext2D 属性

属性名	简要说明
fillStyle	设置填充路径时所用的填充风格，该属性支持三种类型的值： 符合颜色格式的字符串值，表明使用纯色填充 CanvasGeadient，表明使用渐变填充 CanvasPattern，表明是渐变填充
strokeStyle	设置绘制路径时所用的填充风格，该属性支持三种类型的值： 符合颜色格式的字符串值，表明使用纯色填充 CanvasGeadient，表明使用渐变填充 CanvasPattern，表明是渐变填充
Font	设置绘制字符串时所用的字体
globalAlpha	设置全局透明度
globalCompositeOperation	设置全局叠加效果

1、Canvas基础知识

- CanvasRenderingContext2D属性功能用法如下：

表 18-02 CanvasRenderingContext2D 属性

属性名	简要说明
lineCap	设置线段端点的绘图形状。该属性支持如下三个值： “butt”，默认的属性值，该属性值指定不绘制端点，线条结尾处直接结束。 “round”，该属性值指定绘制圆形端点。线条结尾处绘制一个直径为线条宽度的半圆 “square”，该属性值制定绘制正反醒端点。线条结尾处绘制半个边长为线条宽度的正方形。这种形状的端点“butt”形状端点相似，但线条略长。
lineJoin	设置线条连接点的风格。该属性支持如下 3 个值： meter，默认属性值，线条连接点形如箭头 round，线条连接点形如圆角 bevel，线条连接点形如平角
miterLimit	把 lineJoin 树形设置为 meter 风格时，该属性控制锐角箭头的长度
linewidth	设置笔触线条宽度
shadowBlur	设置阴影的模糊程度
shadowColor	设置阴影的颜色
shadowOffsetX	设置阴影在 X 方向上的偏移
shadowOffsetY	设置阴影在 Y 方向上的偏移
textAlign	设置绘制字符串的水平对齐方式，该属性支持 start、end、left、right、center 等属性值
textBaseAlign	设置绘制字符串的垂直对齐方式，该属性支持 top、hanging、middle、alphabetic、idecgraphic、bottom 等属性值

1、Canvas基础知识

案例1:

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <title>第一个Canvas图形</title>
</head>
<body>
    <canvas id="demo" width="300" height="200" style="border:1px solid #CCCCCC;"></canvas>
    <script type="text/javascript">
        //获取canvas元素对应的DOM对象
        var canvas = document.getElementById("demo");
        //获取在canvas上绘图的canvasRenderingContext2D对象
        var ctx = canvas.getContext("2d");
        //设置填充颜色
        ctx.fillStyle = '#007ACC';
        //绘制矩形
        ctx.fillRect(50, 50, 200, 100);
    </script>
</body>
</html>
```

2、图形绘制

- CanvasRenderingContext2D提供了fillRect() 和strokeRect()这2个绘制矩形的方法：
- fillRect(float x, float y, float width, float height)：用于填充一个矩形区域，前两个参数x、y定义该矩形区域的起点坐标，决定了矩形的位置；width定义矩形区域的宽度；height定义矩形区域的高度。
- strokeRect(float x, float y, float width, float height)：用于绘制一个矩形边框，也就是用线条绘制出矩形的轮廓参数，功能和上一个方法相同。

2、图形绘制

案例2:

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <title>绘制简单矩形</title>
</head>
<body>
    <canvas id="demo" width="400" height="180" style="border:1px solid #CCCCCC;"></canvas>
    <script type="text/javascript">
        //获取canvas元素对应的DOM对象
        var canvas = document.getElementById("demo");
        //获取在canvas上绘图的canvasRenderingContext2D对象
        var ctx = canvas.getContext("2d");
        //设置填充颜色
        ctx.fillStyle = '#007ACC';
        //绘制矩形
```

2、图形绘制

- 线条
- 线条在Canvas绘图中被称为路径。
- 在Canvas上使用路径的步骤如下：
- 定义路径，调用CanvasRenderingContext2D对象的beginPath()方法；
- 定义子路径，可以使用的方法有arc()、arcTo()、bezierCurveTo()、lineTo()、moveTo()、quadraticCurveTo()、rect()；
- 关闭路径，调用CanvasRenderingContext2D对象的closePath()方法；
- 填充路径或绘制路径，调用CanvasRenderingContext2D对象的fill()方法或stroke()方法。

2、图形绘制

- 线条
- CanvasRenderingContext2D绘制线条方法介绍如下：
- moveTo (float x, float y)：把Canvas的当前路径结束点移动到x、y对应的点；
- lineTo (float x, float y)：把Canvas的当前路径从当前结束点连接到x、y的对应点。

2、图形绘制

案例3：<body>

```
    <canvas id="demo" width="300" height="200" style="border:1px solid #CCCCCC;"></canvas>
    <script type="text/javascript">
        //获取canvas元素对应的DOM对象
        var canvas = document.getElementById("demo");
        //获取在canvas上绘图的canvasRenderingContext2D对象
        var ctx = canvas.getContext("2d");
        //设置填充颜色
        ctx.strokeStyle = '#007ACC';
        ctx.beginPath(); //开始定义路径
        ctx.moveTo(10, 10); //把Canvas的当前点移动到位置 (x, y)
        ctx.lineTo(290, 190);//把Canvas的当前路径从当前位置连接到 (x、y) 的对应点
        ctx.closePath(); //关闭路径
        //绘制线条路径
        ctx.stroke();
    </script>
</body>
```

2、图形绘制

- 多边形
- CanvasRenderingContext2D只提供了绘制矩形的方法，要使用路径才能绘制复杂的几何图形。
- 正多边形中心点为（dx，dy），外圆半径为size，边数为n，相邻两定点与中心点形成的角的弧度为2 * Math.PI / n。

圆角矩形

- 通过CanvasRenderingContext2D绘制矩形的方法，设置lineJoin = "round"可以向画布添加圆角矩形，但矩形的圆角不可控制。
- CanvasRenderingContext2D使用arcTo()方法绘制可控的圆角矩形。
- arcTo(float x1, float y1, float x2, float y2, float radius)：向Canvas的当前路径上添加一段圆弧。
- arcTo()方法确定一段圆弧的方式是：假设从当前点到P1(x1,y1)绘制一条线段，再从P1(x1,y1)到P2(x2,y2)绘制一条线段，arcTo()则绘制一端同时与上面两条线段相切，且半径为radius的圆弧。

2、图形绘制

2.4圆角矩形

- 圆角矩形

案例5: <canvas id="demo" width="200" height="200" style="border:1px solid #CCCCCC;"></canvas>

```
<script type="text/javascript">
    //获取canvas元素对应的DOM对象
    var canvas = document.getElementById("demo");
    //获取在canvas上绘图的canvasRenderingContext2D对象
    var ctx = canvas.getContext("2d");
    getRroundedRectangle(ctx,10, 100, 100, 20, 20);
    function getRroundedRectangle(context,r, width, height, offsetX, offsetY) {
        //设置线条颜色
        context.strokeStyle = '#007ACC';
        //设置线条宽度
        context.lineWidth = 1;
        context.beginPath();//开始路径
        context.moveTo(offsetX + r, offsetY);
        context.lineTo(offsetX + width - r, offsetY);
        context.arcTo(offsetX + width, offsetY, offsetX + width, offsetY + r, r);
        context.lineTo(offsetX + width, offsetY + height - r);
        context.arcTo(offsetX + width, offsetY + height, offsetX + width - r, offsetY + height, r);
        context.lineTo(offsetX + r, offsetY + height);
        context.arcTo(offsetX, offsetY + height, offsetX, offsetY + height - r, r);
        context.lineTo(offsetX, offsetY + r);
        context.arcTo(offsetX, offsetY, offsetX + r, offsetY, r);
        context.closePath();//结束路径
        context.fillStyle = '#007ACC';
        context.fill();
    }
</script>
```

2、图形绘制

圆形

- 绘制圆形同样需要在Canvas上启用路径，通过路径绘制图形。
- CanvasRenderingContext2D绘制圆形的方法如下：
- arc(float x, float y, float radius, float startAngel, float endAngel, boolen antclockwise)：用于向当前路径添加一段圆弧。
- 圆心坐标为（x，y），半径为radius，开始角度为startAngel，结束角度为endAngel。startAngel、endAngel以为单位，counterclockwise是否为逆时针方向。

2、图形绘制

2.5圆形

- 案例6：<body>
- <canvas id="demo" width="300" height="300" style="border:1px solid #CCCCCC;"></canvas>
- <script type="text/javascript">
- //获取canvas元素对应的DOM对象
- var canvas = document.getElementById("demo");
- //获取在canvas上绘图的canvasRenderingContext2D对象
- var ctx = canvas.getContext("2d");
- //设置线条颜色
- ctx.strokeStyle = '#007ACC';
- //设置线条宽度
- ctx.lineWidth = 1;
- ctx.beginPath();
- ctx.arc(150, 150, 130, 0, Math.PI * 2, true);
- ctx.closePath();
- ctx.stroke();
- //设置填充颜色
- ctx.fillStyle = '#007ACC';
- ctx.beginPath();
- ctx.arc(150, 150, 110, 0, Math.PI * 2, true);
- ctx.closePath();
- ctx.fill();
- </script>
- </body>

曲线

- CanvasRrenderingContext2D提供了bezierCurveTo()和quadraticCurveTo()两个方法，可以向Canvas的当前路径上添加复杂的曲线。
- 两种方法的区别与联系如下：
- bezierCurveTo()和quadraticCurveTo()都是贝塞尔曲线，bezierCurveTo()是一种三次贝塞尔曲线，quadraticCurveTo()是一种二次贝塞尔曲线。

2、图形绘制

- 两种方法的功能和属性用法如下：

- bezierCurveTo(float cpX1, float cpY1, float cpX2, float cpY2, float x, float y)：向Canvas的当前路径添加一段贝塞尔曲线。贝塞尔曲线起点为当前点，终点为（x，y），第一个控制点坐标为（cpX1，cpY1），第二个控制点坐标为(cpX3，cpY2)。

- quadraticCurveTo(float cpX, float cpY, float x, float y)：向Canvas当前路径添加一段二次曲线。

2、图形绘制

文字

- Canvas不仅能绘制图形，还能够显示文本。
- Canvas中的文本是以图像形式绘制的，一旦文字绘制之后，就无法编辑，除非先擦除文字，再重新绘制。
- CanvasRenderingContext2D提供的绘制文字的方法功能用法如下：
- fillText(string text, float x, float y[, float maxWidth])：用于填充字符串。
- strokeText(string text, float x, float y[, float maxWidth]))：用于绘制字符串边框。

2、图形绘制

案例8：绘制文字

```
<body>
    <canvas id="demo" width="400" height="100" style="border:1px solid #CCCCCC;"></canvas>
    <script type="text/javascript">
        // 获取canvas元素对应的DOM对象
        var canvas = document.getElementById('demo');
        // 获取在canvas上绘图的CanvasRenderingContext2D对象
        var ctx = canvas.getContext('2d');
        ctx.fillStyle = "#2B579A";
        ctx.strokeStyle = "#2B579A";
        ctx.font = "bold 20px 微软雅黑";
        ctx.textAlign = "left";
        ctx.textBaseline = "top";
        ctx.fillText("Web前端技术与实践", 30, 10);
        ctx.fillText("Web前端技术与实践", 220, 10,100);
        ctx.strokeText("Web前端技术与实践", 30, 40);
    </script>
</body>
```

2、图形绘制

绘制图像

- 绘制图像时，需使用drawImage方法，CanvasRenderingContext2D为绘位图提供了三种用法：
- void drawImage(image image, x, y)：直接绘制，用于把image绘制到（x，y）处，不会对图片做任何缩放处理，绘制出来的图片保持原来的大小。
- void drawImage(image image, float x, float y, float width, float height)：绘制并指定大小，该方法按照指定大小（width、height）把image绘制到（x，y）处。
- void drawImage(image image, integer sx, integer sy, integer sw, integer sh, float dx, float dy, float dw, float dh)：裁剪图片。
- integer sx, integer sy,裁剪图片的坐标
- integer sw, integer sh被裁剪的宽度和高度
- float dx, float dy放置图片的位置
- float dw, float d图片的高度和宽度

2、图形绘制

案例 1 1 : <body>

```
    <canvas id="demo" width="400" height="200" style="border:1px solid #CCCCCC;"></canvas>
    <script type="text/javascript">
        //获取canvas元素对应的DOM对象
        var canvas = document.getElementById("demo");
        //获取在canvas上绘图的canvasRenderingContext2D对象
        var ctx = canvas.getContext("2d");
        var image = new Image();
        image.src = 'images/html5.png';
        image.onload = function () {
            ctx.drawImage(image, 0, 0);
            ctx.drawImage(image, 126, 0, 100, 100);
            ctx.drawImage(image, 23, 21, 80, 90, 0, 126, 40, 45);
        }

    </script>
</body>
```

2、图形绘制

图像平铺

- 图像平铺就是用图像将画布填满，是绘制图像的一个重要功能。
- 实现平铺技术有两种方法：
- 一种是使用前面所介绍的drawImage()方法。
- 另一种实现平铺效果方法是CanvasRenderingContext2D的createPattern方法。

2、图形绘制

案例12：drawImage平铺

```
<body>
    <canvas id="demo" width="400" height="200" style="border:1px solid #CCCCCC;"></canvas>
    <script type="text/javascript">
        //获取canvas元素对应的DOM对象
        var canvas = document.getElementById("demo");
        //获取在canvas上绘图的canvasRenderingContext2D对象
        var ctx = canvas.getContext("2d");
        var image = new Image();
        image.src = 'images/html5.png';
        image.onload = function () {
            ctx.drawImage(image, 0, 0);
            ctx.drawImage(image, 126, 0, 100, 100);
            ctx.drawImage(image, 23, 21, 80, 90, 0, 126, 40, 45);
        }

    </script>
</body>
```

2、图形绘制

案例13：createPattern平铺

```
<canvas id="demo" width="400" height="200" style="border:1px solid #CCCCCC;"></canvas>
    <script type="text/javascript">
        //获取canvas元素对应的DOM对象
        var canvas = document.getElementById("demo");
        //获取在canvas上绘图的canvasRenderingContext2D对象
        var ctx = canvas.getContext("2d");
        var image = new Image();
        image.src = 'images/html5.png';
        image.onload = function () {
            var pattern = ctx.createPattern(image, 'repeat');//获取平铺对象
            ctx.fillStyle = pattern;//画布填充样式
            ctx.fillRect(0,0,400,200);
        }

    </script>
```

2、图形绘制

- 图像裁剪
- 使用Canvas绘制图像时，经常只需要保留图像的一部分，使用Canvas API自带的图像裁剪功能可以实现这一功能。
- 使用CanvasRenderingContext2D的clip方法实现Canvas元素的图像裁剪功能，具体步骤如下：
- 将需要从图像上裁剪的区域定义成Canvas上的路径。
- 调用ConvasRenderingContext2D的clip()方法把路径裁剪下来。
- 绘制图像，只有被clip()方法裁剪的路径覆盖的部分才会被显示出来。

2、图形绘制

案例 1 4 :

```
<body>
    <canvas id="demo" width="400" height="200" style="border:1px solid #CCCCCC;"></canvas>
    <script type="text/javascript">
        //获取canvas元素对应的DOM对象
        var canvas = document.getElementById("demo");
        //获取在canvas上绘图的canvasRenderingContext2D对象
        var ctx = canvas.getContext("2d");
        var image = new Image();
        image.src = 'images/html5.png';
        var num = 0;
        var item = Math.PI / 6;
        image.onload = function () {
            ctx.beginPath();
            ctx.arc(200, 100, 60, 0, 2*Math.PI, false);
            ctx.lineTo(200, 100);
            ctx.closePath();
            ctx.clip();
            ctx.drawImage(image, 137, 37);
        }
    </script>
</body>
```

2、图形绘制

- 像素处理
- Canvas API能获取图像中的每一个像素，得到该像素的RGBA值。
- 使用图形上下文对象的getImageData方法来获取图像中的像素，该方法的定义如下：
- var imageData = context.getImageData(sx, sy, sw, sh)：sx，sy分别表示获取区域的起点横坐标、起点纵坐标，sw、sh分别表示所获取区域宽度和高度。
- context.putImageData(imagedata, dx, dy[, dirtyX, dirty, dirtyWidth, dirtyHeight])：imageData为前面所述的像素数组，dx、dy分别表示重绘图像的起点横坐标、起点纵坐标。

2、图形绘制

案例15:

```
<body>
    <canvas id="demo" width="400" height="200" style="border:1px solid #CCCCCC;"></canvas>
    <script type="text/javascript">
        //获取canvas元素对应的DOM对象
        var canvas = document.getElementById("demo");
        //获取在canvas上绘图的canvasRenderingContext2D对象
        var ctx = canvas.getContext("2d");
        var image = new Image();
        image.src = 'images/html5.png';
        image.onload = function () {
            ctx.drawImage(image, 137, 37);
            //获取绘制图像数据
            var imageData = ctx.getImageData(137, 37, image.width, image.height);
            console.log(imageData.data);//原图像数据
            for (var i = 0, length = imageData.data.length; i < length; i += 4) {
                imageData.data[i + 3] = imageData.data[i + 3] * 0.5;
            }
            console.log(imageData.data);//修改透明度后图像数据
            //重置图像数据
            ctx.putImageData(imageData, 137, 37);
        }
    </script>
</body>
```

2、图形绘制

- 位图输出
- 当程序CanvasRenderingContext2D通过CanvasRenderingContext2D在Canvas上绘图完成后，通常会需要将该图形或图像输出保存到文件中，可以调用Canvas提供的toDataURL()方法输出位图。
- toDataURL方法的用法如下：
- toDataURL(string type)：该方法把Canvas对应的位图编码成DataURL格式的字符串。其中参数type是一个形如image/png格式的MIME字符串。

2、图形绘制

案例16：

```
<body>
    <canvas id="demo" width="400" height="200" style="border:1px solid #CCCCCC;"></canvas>
    <img id="autoImage" src="" style="display:block;border:1px solid #CCCCCC;"/>
    <script type="text/javascript">
        //获取canvas元素对应的DOM对象
        var canvas = document.getElementById("demo");
        //获取在canvas上绘图的canvasRenderingContext2D对象
        var ctx = canvas.getContext("2d");
        var image = new Image();
        image.src = 'images/html5.png';
        var num = 0;
        var item = Math.PI / 6;
        image.onload = function () {
            ctx.beginPath();
            ctx.arc(200, 100, 60, 0, 2*Math.PI, false);
            ctx.lineTo(200, 100);
            ctx.closePath();
            ctx.clip();
            ctx.drawImage(image, 137, 37);
            document.getElementById("autoImage").src = canvas.toDataURL("image/png");
        }
    </script>
</body>
```

3、图形变换与控制

- 坐标变换
- CanvasRenderingContext2D提供坐标变换支持，通过使用坐标变换，Web前端开发者无须繁琐地计算每个点的坐标，只需对坐标系统进行整体变换即可。
- CanvasRenderingContext2D支持的坐标变换有平移、缩放和旋转三种操作，对应的方法分别为translate()，scale()和rotate()：
- translate(float dx, foat dy)：用作平移坐标系统。
- scale(float sx, float sy)：缩放坐标系统。
- rotate(float angle)：旋转坐标系统。

3、图形变换与控制

- CanvasRenderingContext2D提供两种方法来保存、恢复绘图状态：
- save()：保存当前的绘图状态。
- restore()：恢复之前保存的绘图状态。

3、图形变换与控制

案例17：平移

```
<body>
    <canvas id="demo" width="400" height="200" style="border:1px solid #CCCCCC;"></canvas>
    <script type="text/javascript">
        //获取canvas元素对应的DOM对象
        var canvas = document.getElementById("demo");
        //获取在canvas上绘图的canvasRenderingContext2D对象
        var ctx = canvas.getContext("2d");
        ctx.strokeStyle = "#007ACC";
        drawRound();//绘制圆形边框
        ctx.translate(20, 20);
        drawRound();//绘制圆形边框
        function drawRound() {
            ctx.beginPath();
            ctx.arc(60, 60, 60, 0, 2 * Math.PI, false);
            ctx.closePath();
            ctx.stroke();
        }
    </script>
</body>
```

3、图形变换与控制

案例18：缩放

```
<body>
    <canvas id="demo" width="400" height="200" style="border:1px solid #CCCCCC;"></canvas>
    <script type="text/javascript">
        //获取canvas元素对应的DOM对象
        var canvas = document.getElementById("demo");
        //获取在canvas上绘图的canvasRenderingContext2D对象
        var ctx = canvas.getContext("2d");
        ctx.strokeStyle = "#007ACC";
        drawRound();//绘制圆形边框
        ctx.scale(0.5, 0.5);
        drawRound();//绘制圆形边框
        function drawRound() {
            ctx.beginPath();
            ctx.arc(60, 60, 60, 0, 2 * Math.PI, false);
            ctx.closePath();
            ctx.stroke();
        }
    </script>
</body>
```

3、图形变换与控制

- 案例20：综合

```
<canvas id="demo" width="400" height="350" style="border:1px solid #CCCCCC;"></canvas>
    <script type="text/javascript">
        //获取canvas元素对应的DOM对象
        var canvas = document.getElementById("demo");
        //获取在canvas上绘图的canvasRenderingContext2D对象
        var ctx = canvas.getContext("2d");
        ctx.fillStyle = "#007ACC";
        ctx.translate(30, 200);
        for (var i = 0; i < 30; i++) {
            ctx.translate(50, 50);
            ctx.scale(0.93, 0.93);
            ctx.rotate(-Math.PI / 10);
            ctx.fillRect(0, 0, 75, 75);
        }
    </script>
```

3、图形变换与控制

- 矩阵变换
- 矩阵变换是CanvasRenderingContext2D提供的一个更通用的坐标变换方法transform()。
- 矩阵变换方法的具体使用方法如下：
- transform(m11, m12, m21, m22, dx, dy)：这是一个基于矩阵的变换方法。其中前4个参数组成变换矩阵；dx, dy负责对坐标系统进行平移。

$$\{x, y\} * \begin{Bmatrix} m11 & m12 \\ m21 & m22 \end{Bmatrix} = \{x*m11 + y*m21, x * m12 + y * m22\}$$

3、图形变换与控制

- 设置阴影
- 阴影是图形展示中不可或缺的效果，经常在Web和图形设计中使用。
- 在画布中创建阴影效果是相对较简单的，它可以通过4个全局属性进行控制，具体如下：
- shadowBlur：设置阴影的模糊度。
- shadowColor：设置阴影的颜色。
- shadowOffsetX：设置阴影X方向的偏移。
- shadowOffsetY：设置阴影Y方向的偏移。

3、图形变换与控制

3.3设置阴影

```
案例22: <script type="text/javascript">
        // 获取canvas元素对应的DOM对象
        var canvas = document.getElementById('demo');
        // 获取在canvas上绘图的CanvasRenderingContext2D对象
        var ctx = canvas.getContext('2d');
        ctx.fillStyle = "#2B579A";
        ctx.strokeStyle = "#2B579A";
        ctx.font = "bold 30px 微软雅黑";
        ctx.textAlign = "left";
        ctx.textBaseline = "top";
        ctx.shadowblur = 20;
        ctx.shadowColor = "rgb(0,0,0)";
        ctx.shadowOffsetX = 5;
        ctx.shadowOffsetY = 5;
        ctx.fillText("Web前端技术与实践", 30, 10);
        //透明灰色阴影
        ctx.shadowColor = "rgba(100,100,100,0.5)";
        ctx.fillText("Web前端技术与实践", 30, 50);
        //圆形蓝色阴影
        ctx.shadowColor = "rgb(58,151,255)";
        ctx.beginPath();
        ctx.arc(200, 140, 50, 0, Math.PI * 2, false);
        ctx.closePath();
        ctx.fill();
    </script>
```

3、图形变换与控制

- 叠加风格
- CanvasRebderingContext2D绘图时，后面绘制的图形会默认完全覆盖在前面绘制的图形。
- 特殊情况下需要其他叠加风格，可修改CanvasRebderingContext2D的globalCompositeOperation属性来实现。

3、图形变换与控制

- globalCompositeOperation各属性如表所示：

属性	简要说明
source-over	新绘制的图形将会显示在顶层，覆盖以前绘制的图形。该值为默认值
destination-over	新绘制的图形将放在原图形后面
source-in	新绘制的图形与原图形做 in 运算，只显示新图形与原图形重叠的部分，新图形与原图形的其他部分都变成透明
source-out	新绘制的图形与原图形做 out 运算，只显示新图形与原图形不重叠的部分，新图形与原图形的其他部分都变成透明
destination-out	新绘制的图形与原图形做 out 运算，只显示原图形与新图形不重叠的部分，新图形与原图形的其他部分都变成透明
source-atop	只绘制新图形与原图形重叠部分和原图形未被覆盖部分。新图形的其他部分变为透明
destination-atop	只绘制原图形与新图形重叠部分和新图形未重叠部分。原图形的其他部分变为透明，不绘制新图形的其他部分
ligher	新图形和原图形都绘制。重叠部分绘制两种颜色相加的颜色
xor	绘制新图形与原图形不重叠的部分，重叠部分变成透明的
copy	只绘制新图形，原图形变成透明的。

3、图形变换与控制


```
案例23：  <script type="text/javascript">
        //获取canvas元素对应的DOM对象
        var canvas = document.getElementById("demo");
        //获取在canvas上绘图的canvasRenderingContext2D对象
        var ctx = canvas.getContext("2d");
        overlay("destination-over");
        function overlay(type) {
            //设置填充颜色
            ctx.fillStyle = '#007ACC';
            //绘制矩形
            ctx.fillRect(20, 20, 150, 90);
            ctx.globalCompositeOperation = type;
            //改变填充颜色
            ctx.fillStyle = '#DBEAF9';
            //绘制矩形
            ctx.fillRect(30, 30, 150, 90);
        }
    </script>
```

3、图形变换与控制

- 线性渐变
- 线性渐变方法的具体使用方法如下所示：
- `createLinearGradient(float xStart, float yStart, float xEnd, float yEnd)`：四个参数分别表示渐变开始横坐标、渐变开始纵坐标、渐变结束横坐标、渐变结束纵坐标。

3、图形变换与控制

- 线性渐变
- 线性渐变使用步骤如下所示：
- 调用CanvasRenderingContext2D的createLinearGradient(float xStart, float yStart, float xEnd, float yEnd)方法创建一个线性渐变，该方法返回一个CanvasGradient对象。
- 调用CanvasGradient对象的addColorStop(float offset, string color)方法向线性渐变中添加颜色。
- 将CanvasGradient对象赋值给CanvasRenderingContext2D的fillStyle或strokeStyle属性。

3、图形变换与控制

- 圆形渐变
- 圆形渐变使用createRadialGradient()方法。
- createRadialGradient(float xstart, float ystart, float radiusStart, float xEnd, float yEnd, float radiusEnd):
- xStart、yStart控制渐变开始的圆圈圆心。
- radiusStart控制开始圆圈的半径。
- xEnd、yEnd控制渐变结束圆圈的圆心。
- radiusEnd控制结束圆圈的半径。

3、图形变换与控制

- 位图填充
- Canvas提供了CanvasPattern对象用于实现位图填充，位图填充方式有填充背景和填充边框2种：
- 填充背景已经在前面使用createPattern()方法实现图像平铺中应用；
- CanvasPattern对象既可赋值给strokeStyle属性作为（作为几何形状的边框），也可以赋值给fillStyle树形（作为集合形状的填充）。

3、图形变换与控制

- 案例18-24：线性渐变
- 案例18-25：圆形渐变
- 案例18-26：位图填充