

Canvas API



- [Canvas API](#)
- [Canvas API](#)
 - [Canvas API](#)
 - [Canvas API](#)
 - [Canvas API](#)
 - [Canvas API](#)
 - [Canvas API](#)
 - [Canvas API](#)
 - [Canvas API](#)
 - [Canvas API](#)
- [Canvas API](#)
 - [CanvasRenderingContext2D.drawImage\(\)](#)
 - [CanvasRenderingContext2D.save\(\)](#) [CanvasRenderingContext2D.restore\(\)](#)
 - [CanvasRenderingContext2D.canvas](#)
 - [CanvasRenderingContext2D](#)
- [<canvas>](#) [HTMLCanvasElement](#)
 - [HTMLCanvasElement.toDataURL\(\)](#)
 - [HTMLCanvasElement.toBlob\(\)](#)
- [Canvas](#) [Canvas](#)
 - [Canvas](#)
 - [Canvas](#)
- [Canvas](#)



The diagram shows a light blue rounded rectangle labeled `<canvas>` on the left. To its right is a 2D array structure consisting of two rows of 10 empty square cells each, representing a 2x10 grid.

JavaScript

API
JavaScript

bitmap

- `CanvasRenderingContext2D.fill()`
- `CanvasRenderingContext2D.stroke()`
- `CanvasRenderingContext2D.fillStyle`
- `CanvasRenderingContext2D.strokeStyle`

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');
```

```
ctx.beginPath();
ctx.moveTo(100, 100);
ctx.lineTo(200, 200);
ctx.lineTo(100, 200);
```

```
ctx.fill()
// 
ctx.stroke()
```

`fill()`

`stroke()`

`fillStyle` `strokeStyle`

```
ctx.fillStyle = 'red';
ctx.fill();
// 
ctx.strokeStyle = 'red';
ctx.stroke();
```



- `CanvasRenderingContext2D.lineWidth` 1.0
- `CanvasRenderingContext2D.lineCap` `butt`
 `round` `square`
- `CanvasRenderingContext2D.lineJoin` `round`
 `bevel` `miter`)

- `CanvasRenderingContext2D.miterLimit` `10` `lineJoin`
- `CanvasRenderingContext2D.miter`
- `CanvasRenderingContext2D.getLineDash()`
- `CanvasRenderingContext2D.setLineDash()`

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');
```

```
ctx.beginPath();
ctx.moveTo(100, 100);
ctx.lineTo(200, 200);
ctx.lineTo(100, 200);

ctx.lineWidth = 3;
ctx.lineCap = 'round';
ctx.lineJoin = 'round';
ctx.setLineDash([15, 5]);
ctx.stroke();
```

`3`



- `CanvasRenderingContext2D.rect()`
- `CanvasRenderingContext2D.fillRect()`
- `CanvasRenderingContext2D.strokeRect()`
- `CanvasRenderingContext2D.clearRect()`

`CanvasRenderingContext2D.rect()`

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');

ctx.rect(10, 10, 100, 100);
ctx.fill();
```

`(10, 10)` `100`

CanvasRenderingContext2D.fillRect() 

```
var canvas = document.getElementById('myCanvas');  
var ctx = canvas.getContext('2d');  
  
ctx.fillStyle = 'green';  
ctx.fillRect(10, 10, 100, 100);
```



(10, 10) 

100 

CanvasRenderingContext2D.strokeRect() 

```
var canvas = document.getElementById('myCanvas');  
var ctx = canvas.getContext('2d');  
  
ctx.strokeStyle = 'green';  
ctx.strokeRect(10, 10, 100, 100);
```



(10, 10) 

100 

CanvasRenderingContext2D.clearRect() 



```
var canvas = document.getElementById('myCanvas');  
var ctx = canvas.getContext('2d');  
  
ctx.fillRect(10, 10, 100, 100);  
ctx.clearRect(15, 15, 90, 90);
```



100 x 100 

90 x 90 

5







- CanvasRenderingContext2D.arc() 
- CanvasRenderingContext2D.arcTo() 

CanvasRenderingContext2D.arc() 

```
// 画圆
ctx.arc(x, y, radius, startAngle, endAngle, anticlockwise)

// 画圆
ctx.arc(5, 5, 5, 0, 2 * Math.PI, true)
```

```
arc() 画圆 x y 圆心的坐标 radius 圆的半径 startAngle 开始角度 endAngle 结束角度
anticlockwise 是否逆时针 true 是否填充
false 是否填充
```

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');

ctx.beginPath();
ctx.arc(60, 60, 50, 0, Math.PI * 2, true);
ctx.fill();
```

```
50 圆的半径 0 开始角度 2 * PI 结束角度
```

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');

ctx.beginPath();
ctx.moveTo(50, 20);
ctx.arc(100, 20, 50, 0, Math.PI, false);
ctx.stroke();
```

```
CanvasRenderingContext2D.arcTo()
画圆弧
```

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');

ctx.beginPath();
ctx.moveTo(0, 0);
ctx.arcTo(50, 50, 100, 0, 25);
ctx.lineTo(100, 0);
```

```
ctx.stroke();
```

```
arcTo() 5
(0, 0) (50, 50) (50, 50) (100, 0)
```



- CanvasRenderingContext2D.fillText()
- CanvasRenderingContext2D.strokeText()
- CanvasRenderingContext2D.measureText() TextMetrics
- CanvasRenderingContext2D.font 10px sans-serif
- CanvasRenderingContext2D.textAlign start
- CanvasRenderingContext2D.direction inherit
- CanvasRenderingContext2D.textBaseline alphabetic

```
fillText() 
```

```
CanvasRenderingContext2D.fillText(text, x, y [, maxWidth])
```

- text
- x
- y
- maxWidth

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');

ctx.fillText('Hello world', 50, 50);
```

```
(50, 50) Hello world
```

```
fillText() 
```

```
fillText() 
```

```
strokeText() 
```

```
fillText() 
```

```
var canvas = document.getElementById('myCanvas');  
var ctx = canvas.getContext('2d');  
  
ctx.strokeText('Hello world', 50, 50);
```

Diagram illustrating the relationship between CSS font properties and their visual representation:

- Top row: A series of 10px square boxes, followed by a box labeled "10px", a box labeled "sans-serif", and a box labeled "font".
- Bottom row: A series of 10px square boxes, followed by a box labeled "CSS", a box labeled "font", and a series of 10px square boxes.

```
var canvas = document.getElementById('myCanvas');  
var ctx = canvas.getContext('2d');  
  
ctx.font = 'Bold 20px Arial';  
ctx.fillText('Hello world', 50, 50);
```

`textAlign`

- left 
- right 
- center 
- start 
- end 

```
var canvas = document.getElementById('myCanvas');  
var ctx = canvas.getContext('2d');  
  
ctx.font = 'Bold 20px Arial';  
ctx.textAlign = 'center';  
ctx.fillText('Hello world', 50, 50);
```

direction inherit <canvas> document
ltr rtl

textBaseline

- top 
- hanging 
- middle 
- alphabetic 
- ideographic 
- bottom   ideographic 

measureText()  TextMetrics

 width 

□□□□ 10px □□□□ Hello world □□□□ 49.46 □□□ 20px □□□□ 107.78 □



- ```
createLinearGradient()
```

[illegible]

The diagram illustrates the mapping of CSS color values to CanvasGradient and addColorStop() calls. It shows three rows of color values: a row of 10 empty boxes, a row of 10 empty boxes, and a row of 10 empty boxes. The first row is labeled 'CanvasGradient' and the second row is labeled 'addColorStop()'. The third row shows the mapping of color values to the addColorStop() call, with '0' and '1' indicating the position of the color stop. The label 'CSS' is positioned below the third row.

 `gradient`  `fillStyle`  `fillRect()`



createRadialGradient()

```
ctx.createRadialGradient(x0, y0, r0, x1, y1, r1)
```

```
createRadialGradient() [] [] x0 [] y0 [] r0 [] x1 []
[] y1 [] r1 []
```

 CanvasGradient 

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');

var gradient = ctx.createRadialGradient(100, 100, 50, 100, 100, 100);
gradient.addColorStop(0, 'white');
gradient.addColorStop(1, 'green');
ctx.fillStyle = gradient;
ctx.fillRect(0, 0, 200, 200);
```

```
createPattern()
```

```
ctx.createPattern(image, repetition)
```

The diagram illustrates a 2D array structure. It consists of four rows of boxes representing array elements. The first row has 15 boxes, the second has 10, the third has 10, and the fourth has 15. Labels and dimensions are placed around the array:

- Top right: `<img>` (above a 5-box segment) and `<canvas>` (above a 5-box segment).
- Second row: `Blob` is centered above the first 10 boxes. `repeat` is to the right of the last 5 boxes.
- Third row: `repeat-x ( )` is above the first 5 boxes, `repeat-y ( )` is above the next 5 boxes, and `no-repeat ( )` is above the last 5 boxes.
- Fourth row: `null` is above the first 5 boxes and `null` is above the next 5 boxes.

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');

var img = new Image();
img.src = 'https://example.com/pattern.png';
img.onload = function() {
 var pattern = ctx.createPattern(img, 'repeat');
 ctx.fillStyle = pattern;
 ctx.fillRect(0, 0, 400, 400);
};
```

```
createPattern()
```

- image[ ] [ ] [ ] [ ] [ ]

- sx 
- sy 
- sWidth   

- sHeight   

- dx 
- dy 
- dWidth 
- dHeight 

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');

var img = new Image();
img.src = 'image.png';
img.onload = function () {
 ctx.drawImage(img, 0, 0);
};
```

[illegible]

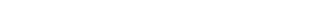
- ```
1 imageData()
```

```
ctx.getImageData(sx, sy, sw, sh)
```

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');

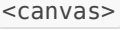
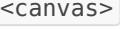
var imageData = ctx.getImageData(0, 0, canvas.width, canvas.height);
```

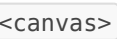
- `ImageData.data`  alpha


- `ImageData.width`  `ImageData` 
- `ImageData.height`  `ImageData` 

2 putImageData()

[illegible]

- imageData  ImageData 
- dx  ImageData 
- dy  ImageData 
- dirtyX  
- dirtyY  
- dirtyWidth  ImageData 
- dirtyHeight  ImageData 

 ImageData  

```
ctx.putImageData(imageData, 0, 0);
```

3 createImageData()

CanvasRenderingContext2D.createImageData()  ImageData 
 0 

```
ctx.createImageData(width, height)
ctx.createImageData(imageData)
```

createImageData() 

- width  ImageData 
- height  ImageData 
- imageData  ImageData 

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');

var imageData = ctx.createImageData(100, 100);
```

  100 x 100 

CanvasRenderingContext2D.save() CanvasRenderingContext2D.restore()

CanvasRenderingContext2D.save()


```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');

ctx.save();
```

|||||

save() |||

CanvasRenderingContext2D.restore()

|||||

|||||

restore |||

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');

ctx.save();

ctx.fillStyle = 'green';
ctx.restore();

ctx.fillRect(10, 10, 100, 100);
```

|||||

restore()

|||||

||

CanvasRenderingContext2D.canvas

CanvasRenderingContext2D.canvas |||

CanvasRenderingContext2D |||

<canvas>

|||||

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');

ctx.canvas === canvas // true
```

||||

|||||

- CanvasRenderingContext2D.rotate() |||

- `CanvasRenderingContext2D.scale()`
- `CanvasRenderingContext2D.translate()`
- `CanvasRenderingContext2D.transform()`
- `CanvasRenderingContext2D.setTransform()`

1 rotate()

```
CanvasRenderingContext2D.rotate()
```

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');
```

```
ctx.rotate(45 * Math.PI / 180);
ctx.fillRect(70, 0, 100, 30);
```

The diagram illustrates the sequence of drawing commands for a rectangle. It shows a series of rectangles being drawn, with the final one being a 45-degree rotated rectangle. The commands are: `fillRect()`, `rotate()`, `translate()`, and `fillRect()`.

2 scale()

The diagram illustrates the `CanvasRenderingContext2D.scale()` method. It shows three arguments: `x` (represented by a box with 8 segments), `y` (represented by a box with 8 segments), and `0.5` (represented by a box with 8 segments). The `0.5` argument is highlighted with a red border. Below the arguments, there are three boxes: the first contains `50%` and the second contains `10`, both with red borders. The third box is empty.

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');
```

```
ctx.scale(10, 3);
ctx.fillRect(10, 10, 10, 10);
```

Diagram illustrating the scaling of a 2D grid (10x10) to a 100x30 grid. The original grid is shown as a 10x10 array of squares. The scaled grid is shown as a 100x30 array of squares. The scaling is performed using the `ctx.scale(1, -1)` method, which scales the grid by a factor of 1 horizontally and -1 vertically. The resulting grid is shown as a 100x30 array of squares, with the original grid's dimensions (10x10) and the scaled grid's dimensions (100x30) indicated.

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');
```

```
ctx.scale(1, -2);
ctx.font = "16px serif";
ctx.fillText('Hello world!', 20, -20);
```



2 Hello World!



3 translate()

`CanvasRenderingContext2D.translate()`

x y



```
var canvas = document.getElementById('myCanvas');  
var ctx = canvas.getContext('2d');  
  
ctx.translate(50, 50);  
ctx.fillRect(0, 0, 100, 100);
```

4 transform()

`CanvasRenderingContext2D.transform()`



```
ctx.transform(a, b, c, d, e, f);  
/*  
a:  (00010000)  
b:  (00000000)  
c:  (00000000)  
d:  (00010000)  
e:  (00000000)  
f:  (00000000)  
*/
```



```
var canvas = document.getElementById('myCanvas');  
var ctx = canvas.getContext('2d');  
  
ctx.transform(2, 0, 0, 1, 50, 50);  
ctx.fillRect(0, 0, 100, 100);
```



100 x 100

200 x 100

(0, 0)

(50, 50)

transform()

5 setTransform()

CanvasRenderingContext2D.setTransform()

transform()

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');

ctx.translate(50, 50);
ctx.fillRect(0, 0, 100, 100);

ctx.setTransform(1, 0, 0, 1, 0, 0);
ctx.fillRect(0, 0, 100, 100);
```

fillRect() (0, 0) (50, 50) setTransform() 1, 0, 0, 1, 0, 0 (0, 0)

<canvas>

CanvasRenderingContext2D <canvas>

HTMLCanvasElement.toDataURL()

<canvas> toDataURL() Canvas Data URI

```
canvas.toDataURL(type, quality)
```

toDataURL()

- type image/png image/jpeg Chrome image/webp
- quality 0 1 JPEG WebP 0.92

Data URI

```
function convertCanvasToImage(canvas) {
  var image = new Image();
  image.src = canvas.toDataURL('image/png');
```

```
return image;
}
```

□□□□□

<canvas> □□□□□

PNG Data URI□

```
var fullQuality = canvas.toDataURL('image/jpeg', 0.9);
var mediumQuality = canvas.toDataURL('image/jpeg', 0.6);
var lowQuality = canvas.toDataURL('image/jpeg', 0.3);
```

□□□□

<canvas> □□□□□□□□□□□□□□

JPEG □□□

HTMLCanvasElement.toBlob()

HTMLCanvasElement.toBlob() □□□□

<canvas> □□□□□

Blob □□□□□□□

image/png

□□□□□□□□

```
// □□
canvas.toBlob(callback, mimeType, quality)

// □□
canvas.toBlob(function (blob) {...}, 'image/jpeg', 0.95)
```

toBlob() □□□□□□□□□□

- callback□□□□□□□□□□ Blob □□□□□
- mimeType□□□□□□ MIMeType □□□□□ image/png □
- quality□□□□ 0□ 1□□□□□□□□□□ image/jpeg □ image/webp □□□□□□□

□□□□□□□□□□

□□□□□

<canvas> □□□□□

 □□□

```
var canvas = document.getElementById('myCanvas');

function blobToImg(blob) {
  var newImg = document.createElement('img');
  var url = URL.createObjectURL(blob);

  newImg.onload = function () {
    // □□□□□□□□ URL □□
    URL.revokeObjectURL(url);
  }
}
```

```
};

newImg.src = url;
document.body.appendChild(newImg);
}

canvas.toBlob(blobToImg);
```

Canvas



Canvas 

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');

var posX = 20;
var posY = 100;

setInterval(function () {
    ctx.fillStyle = 'black';
    ctx.fillRect(0, 0, canvas.width, canvas.height);

    posX += 1;
    posY += 0.25;

    ctx.beginPath();
    ctx.fillStyle = 'white';

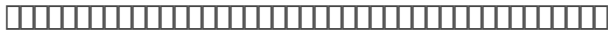
    ctx.arc(posX, posY, 10, 0, Math.PI * 2, true);
    ctx.closePath();
    ctx.fill();
}, 30);
```



30

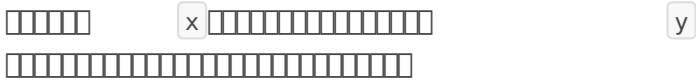
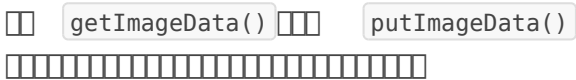
setInterval()





```
var vx = 10;
var vy = -10;
var gravity = 1;

setInterval(function () {
    posX += vx;
    posY += vy;
    vy += gravity;
    // ...
});
```



```
if (canvas.width > 0 && canvas.height > 0) {
    var imageData = context.getImageData(0, 0, canvas.width, canvas.height);
    filter(imageData);
    context.putImageData(imageData, 0, 0);
}
```



```
grayscale = function (pixels) {  
  var d = pixels.data;  
  for (var i = 0; i < d.length; i += 4) {  
    var r = d[i];  
    var g = d[i + 1];  
    var b = d[i + 2];
```

```

    d[i] = d[i + 1] = d[i + 2] = (r + g + b) / 3;
  }
  return pixels;
};

```

```

sepia = function (pixels) {
  var d = pixels.data;
  for (var i = 0; i < d.length; i += 4) {
    var r = d[i];
    var g = d[i + 1];
    var b = d[i + 2];
    d[i]      = (r * 0.393) + (g * 0.769) + (b * 0.189); // red
    d[i + 1] = (r * 0.349) + (g * 0.686) + (b * 0.168); // green
    d[i + 2] = (r * 0.272) + (g * 0.534) + (b * 0.131); // blue
  }
  return pixels;
};

```

```

var red = function (pixels) {
  var d = pixels.data;
  for (var i = 0; i < d.length; i += 4) {
    var r = d[i];
    var g = d[i + 1];
    var b = d[i + 2];
    d[i] = (r + g + b)/3;          // 平均值
    d[i + 1] = d[i + 2] = 0; // 其他两个通道置0
  }
  return pixels;
};

```

4

brightness



```
var brightness = function (pixels, delta) {  
  var d = pixels.data;  
  for (var i = 0; i < d.length; i += 4) {  
    d[i] += delta;    // red  
    d[i + 1] += delta; // green  
    d[i + 2] += delta; // blue  
  }  
  return pixels;  
};
```

5

invert

255 -



```
invert = function (pixels) {  
  var d = pixels.data;  
  for (var i = 0; i < d.length; i += 4) {  
    d[i] = 255 - d[i];  
    d[i + 1] = 255 - d[i + 1];  
    d[i + 2] = 255 - d[i + 2];  
  }  
  return pixels;  
};
```



- David Walsh, [JavaScript Canvas Image Conversion](#)
- Matt West, [Getting Started With The Canvas API](#)
- John Robinson, [How You Can Do Cool Image Effects Using HTML5 Canvas](#)
- Iwaylo Gerchev, [HTML5 Canvas Tutorial: An Introduction](#)
- Donovan Hutchinson, [Particles in canvas](#)

