### **HarmonyOS/OpenHarmony应用开发-ArkTS画布组件CanvasRenderingContext2D对象（九）fillRect strokeRect**

### **fillRect**

fillRect(x: number, y: number, w: number, h: number): void

填充一个矩形。

****参数：****

| **参数** | **类型** | **必填** | **默认值** | **说明** |
| --- | --- | --- | --- | --- |
| x | number | 是 | 0 | 指定矩形左上角点的x坐标。 |
| y | number | 是 | 0 | 指定矩形左上角点的y坐标。 |
| width | number | 是 | 0 | 指定矩形的宽度。 |
| height | number | 是 | 0 | 指定矩形的高度。 |

****示例：****

1. // xxx.ets
2. @Entry
3. @Component
4. struct FillRect {
5. private settings: RenderingContextSettings = new RenderingContextSettings(true)
6. private context: CanvasRenderingContext2D = new CanvasRenderingContext2D(this.settings)
7.
8. build() {
9. Flex({ direction: FlexDirection.Column, alignItems: ItemAlign.Center, justifyContent: FlexAlign.Center }) {
10. Canvas(this.context)
11. .width('100%')
12. .height('100%')
13. .backgroundColor('#ffff00')
14. .onReady(() =>{
15. this.context.fillRect(0,30,100,100)
16. })
17. }
18. .width('100%')
19. .height('100%')
20. }
21. }



### **strokeRect**

strokeRect(x: number, y: number, w: number, h: number): void

绘制具有边框的矩形，矩形内部不填充。

****参数：****

| **参数** | **类型** | **必填** | **默认值** | **说明** |
| --- | --- | --- | --- | --- |
| x | number | 是 | 0 | 指定矩形的左上角x坐标。 |
| y | number | 是 | 0 | 指定矩形的左上角y坐标。 |
| w | number | 是 | 0 | 指定矩形的宽度。 |
| h | number | 是 | 0 | 指定矩形的高度。 |

****示例：****

1. // xxx.ets
2. @Entry
3. @Component
4. struct StrokeRect {
5. private settings: RenderingContextSettings = new RenderingContextSettings(true)
6. private context: CanvasRenderingContext2D = new CanvasRenderingContext2D(this.settings)
7. build() {
8. Flex({ direction: FlexDirection.Column, alignItems: ItemAlign.Center, justifyContent: FlexAlign.Center }) {
9. Canvas(this.context)
10. .width('100%')
11. .height('100%')
12. .backgroundColor('#ffff00')
13. .onReady(() =>{
14. this.context.strokeRect(30, 30, 200, 150)
15. })
16. }
17. .width('100%')
18. .height('100%')
19. }
20. }



### **HarmonyOS/OpenHarmony应用开发-ArkTS画布组件CanvasRenderingContext2D对象（十一）measureText stroke**

### **measureText**

measureText(text: string): TextMetrics

该方法返回一个文本测算的对象，通过该对象可以获取指定文本的宽度值。

****参数：****

| **参数** | **类型** | **必填** | **默认值** | **说明** |
| --- | --- | --- | --- | --- |
| text | string | 是 | '' | 需要进行测量的文本。 |

****返回值：****

|  |  |
| --- | --- |
| **类型** | **说明** |
| TextMetrics | 文本的尺寸信息。 |

****TextMetrics类型描述:****

| **属性** | **类型** | **描述** |
| --- | --- | --- |
| width | number | 字符串的宽度。 |
| height | number | 字符串的高度。 |
| actualBoundingBoxAscent | number | 从CanvasRenderingContext2D.textBaseline 属性标明的水平线到渲染文本的矩形边界顶部的距离，当前值为0。 |
| actualBoundingBoxDescent | number | 从CanvasRenderingContext2D.textBaseline 属性标明的水平线到渲染文本的矩形边界底部的距离，当前值为0。 |
| actualBoundingBoxLeft | number | 平行于基线，从CanvasRenderingContext2D.textAlign 属性确定的对齐点到文本矩形边界左侧的距离，当前值为0。 |
| actualBoundingBoxRight | number | 平行于基线，从CanvasRenderingContext2D.textAlign 属性确定的对齐点到文本矩形边界右侧的距离，当前值为0。 |
| alphabeticBaseline | number | 从CanvasRenderingContext2D.textBaseline 属性标明的水平线到线框的 alphabetic 基线的距离，当前值为0。 |
| emHeightAscent | number | 从CanvasRenderingContext2D.textBaseline 属性标明的水平线到线框中 em 方块顶部的距离，当前值为0。 |
| emHeightDescent | number | 从CanvasRenderingContext2D.textBaseline 属性标明的水平线到线框中 em 方块底部的距离，当前值为0。 |
| fontBoundingBoxAscent | number | 从CanvasRenderingContext2D.textBaseline 属性标明的水平线到渲染文本的所有字体的矩形最高边界顶部的距离，当前值为0。 |
| fontBoundingBoxDescent | number | 从CanvasRenderingContext2D.textBaseline 属性标明的水平线到渲染文本的所有字体的矩形边界最底部的距离，当前值为0。 |
| hangingBaseline | number | 从CanvasRenderingContext2D.textBaseline 属性标明的水平线到线框的 hanging 基线的距离，当前值为0。 |
| ideographicBaseline | number | 从CanvasRenderingContext2D.textBaseline 属性标明的水平线到线框的 ideographic 基线的距离，当前值为0。 |

****示例：****

1. // xxx.ets
2. @Entry
3. @Component
4. struct MeasureText {
5. private settings: RenderingContextSettings = new RenderingContextSettings(true)
6. private context: CanvasRenderingContext2D = new CanvasRenderingContext2D(this.settings)
7. build() {
8. Flex({ direction: FlexDirection.Column, alignItems: ItemAlign.Center, justifyContent: FlexAlign.Center }) {
9. Canvas(this.context)
10. .width('100%')
11. .height('100%')
12. .backgroundColor('#ffff00')
13. .onReady(() =>{
14. this.context.font = '50px sans-serif'
15. this.context.fillText("Hello World!", 20, 100)
16. this.context.fillText("width:" + this.context.measureText("Hello World!").width, 20, 200)
17. })
18. }
19. .width('100%')
20. .height('100%')
21. }
22. }



### **stroke**

stroke(path?: Path2D): void

进行边框绘制操作。

****参数：****

| **参数** | **类型** | **必填** | **默认值** | **描述** |
| --- | --- | --- | --- | --- |
| path | [Path2D](https://developer.harmonyos.com/cn/docs/documentation/doc-references-V3/ts-components-canvas-path2d-0000001428061772-V3) | 否 | null | 需要绘制的Path2D。 |

****示例：****

1. // xxx.ets
2. @Entry
3. @Component
4. struct Stroke {
5. private settings: RenderingContextSettings = new RenderingContextSettings(true)
6. private context: CanvasRenderingContext2D = new CanvasRenderingContext2D(this.settings)
7. build() {
8. Flex({ direction: FlexDirection.Column, alignItems: ItemAlign.Center, justifyContent: FlexAlign.Center }) {
9. Canvas(this.context)
10. .width('100%')
11. .height('100%')
12. .backgroundColor('#ffff00')
13. .onReady(() =>{
14. this.context.moveTo(25, 25)
15. this.context.lineTo(25, 105)
16. this.context.lineTo(75, 105)
17. this.context.lineTo(75, 25)
18. this.context.strokeStyle = 'rgb(0,0,255)'
19. this.context.stroke()
20. })
21. }
22. .width('100%')
23. .height('100%')
24. }
25. }



### **HarmonyOS/OpenHarmony应用开发-ArkTS画布组件CanvasRenderingContext2D对象（十二）beginPath moveTo**

### **beginPath**

beginPath(): void

创建一个新的绘制路径。

****示例：****

1. // xxx.ets
2. @Entry
3. @Component
4. struct BeginPath {
5. private settings: RenderingContextSettings = new RenderingContextSettings(true)
6. private context: CanvasRenderingContext2D = new CanvasRenderingContext2D(this.settings)
7. build() {
8. Flex({ direction: FlexDirection.Column, alignItems: ItemAlign.Center, justifyContent: FlexAlign.Center }) {
9. Canvas(this.context)
10. .width('100%')
11. .height('100%')
12. .backgroundColor('#ffff00')
13. .onReady(() =>{
14. this.context.beginPath()
15. this.context.lineWidth = 6
16. this.context.strokeStyle = '#0000ff'
17. this.context.moveTo(15, 80)
18. this.context.lineTo(280, 160)
19. this.context.stroke()
20. })
21. }
22. .width('100%')
23. .height('100%')
24. }
25. }



### **moveTo**

moveTo(x: number, y: number): void

路径从当前点移动到指定点。

****参数：****

| **参数** | **类型** | **必填** | **默认值** | **说明** |
| --- | --- | --- | --- | --- |
| x | number | 是 | 0 | 指定位置的x坐标。 |
| y | number | 是 | 0 | 指定位置的y坐标。 |

****示例：****

1. // xxx.ets
2. @Entry
3. @Component
4. struct MoveTo {
5. private settings: RenderingContextSettings = new RenderingContextSettings(true)
6. private context: CanvasRenderingContext2D = new CanvasRenderingContext2D(this.settings)
7. build() {
8. Flex({ direction: FlexDirection.Column, alignItems: ItemAlign.Center, justifyContent: FlexAlign.Center }) {
9. Canvas(this.context)
10. .width('100%')
11. .height('100%')
12. .backgroundColor('#ffff00')
13. .onReady(() =>{
14. this.context.beginPath()
15. this.context.moveTo(10, 10)
16. this.context.lineTo(280, 160)
17. this.context.stroke()
18. })
19. }
20. .width('100%')
21. .height('100%')
22. }
23. }



### **HarmonyOS/OpenHarmony应用开发-ArkTS画布组件CanvasRenderingContext2D对象（十三）lineTo closePath**

### **lineTo**

lineTo(x: number, y: number): void

从当前点到指定点进行路径连接。

****参数：****

| **参数** | **类型** | **必填** | **默认值** | **描述** |
| --- | --- | --- | --- | --- |
| x | number | 是 | 0 | 指定位置的x坐标。 |
| y | number | 是 | 0 | 指定位置的y坐标。 |

****示例：****

1. // xxx.ets
2. @Entry
3. @Component
4. struct LineTo {
5. private settings: RenderingContextSettings = new RenderingContextSettings(true)
6. private context: CanvasRenderingContext2D = new CanvasRenderingContext2D(this.settings)
7. build() {
8. Flex({ direction: FlexDirection.Column, alignItems: ItemAlign.Center, justifyContent: FlexAlign.Center }) {
9. Canvas(this.context)
10. .width('100%')
11. .height('100%')
12. .backgroundColor('#ffff00')
13. .onReady(() =>{
14. this.context.beginPath()
15. this.context.moveTo(10, 10)
16. this.context.lineTo(280, 160)
17. this.context.stroke()
18. })
19. }
20. .width('100%')
21. .height('100%')
22. }
23. }



### **closePath**

closePath(): void

结束当前路径形成一个封闭路径。

****示例：****

1. // xxx.ets
2. @Entry
3. @Component
4. struct ClosePath {
5. private settings: RenderingContextSettings = new RenderingContextSettings(true)
6. private context: CanvasRenderingContext2D = new CanvasRenderingContext2D(this.settings)
7. build() {
8. Flex({ direction: FlexDirection.Column, alignItems: ItemAlign.Center, justifyContent: FlexAlign.Center }) {
9. Canvas(this.context)
10. .width('100%')
11. .height('100%')
12. .backgroundColor('#ffff00')
13. .onReady(() =>{
14. this.context.beginPath()
15. this.context.moveTo(30, 30)
16. this.context.lineTo(110, 30)
17. this.context.lineTo(70, 90)
18. this.context.closePath()
19. this.context.stroke()
20. })
21. }
22. .width('100%')
23. .height('100%')
24. }
25. }



### **HarmonyOS/OpenHarmony应用开发-ArkTS画布组件CanvasRenderingContext2D对象（十四）createPattern bezierCurveTo**

### **createPattern**

createPattern(image: ImageBitmap, repetition: string | null): CanvasPattern | null

通过指定图像和重复方式创建图片填充的模板。

****参数：****

| **参数** | **类型** | **必填** | **描述** |
| --- | --- | --- | --- |
| image | [ImageBitmap](https://developer.harmonyos.com/cn/docs/documentation/doc-references-V3/ts-components-canvas-imagebitmap-0000001478061729-V3) | 是 | 图源对象，具体参考ImageBitmap对象。 |
| repetition | string | 是 | 设置图像重复的方式，取值为：'repeat'、'repeat-x'、&nbsp'repeat-y'、'no-repeat'。默认值：'' |

****返回值：****：

|  |  |
| --- | --- |
| **类型** | **说明** |
| [CanvasPattern](https://developer.harmonyos.com/cn/docs/documentation/doc-references-V3/ts-canvasrenderingcontext2d-0000001478181441-V3%22%20%5Cl%20%22ZH-CN_TOPIC_0000001478181441__canvaspattern) | 通过指定图像和重复方式创建图片填充的模板对象。 |

****示例：****

1. // xxx.ets
2. @Entry
3. @Component
4. struct CreatePattern {
5. private settings: RenderingContextSettings = new RenderingContextSettings(true)
6. private context: CanvasRenderingContext2D = new CanvasRenderingContext2D(this.settings)
7. private img:ImageBitmap = new ImageBitmap("common/images/icon.jpg")
8. build() {
9. Flex({ direction: FlexDirection.Column, alignItems: ItemAlign.Center, justifyContent: FlexAlign.Center }) {
10. Canvas(this.context)
11. .width('100%')
12. .height('100%')
13. .backgroundColor('#ffff00')
14. .onReady(() =>{
15. var pattern = this.context.createPattern(this.img, 'repeat')
16. this.context.fillStyle = pattern
17. this.context.fillRect(0, 0, 200, 200)
18. })
19. }
20. .width('100%')
21. .height('100%')
22. }
23. }



### **bezierCurveTo**

bezierCurveTo(cp1x: number, cp1y: number, cp2x: number, cp2y: number, x: number, y: number): void

创建三次贝赛尔曲线的路径。

****参数：****

| **参数** | **类型** | **必填** | **默认值** | **描述** |
| --- | --- | --- | --- | --- |
| cp1x | number | 是 | 0 | 第一个贝塞尔参数的x坐标值。 |
| cp1y | number | 是 | 0 | 第一个贝塞尔参数的y坐标值。 |
| cp2x | number | 是 | 0 | 第二个贝塞尔参数的x坐标值。 |
| cp2y | number | 是 | 0 | 第二个贝塞尔参数的y坐标值。 |
| x | number | 是 | 0 | 路径结束时的x坐标值。 |
| y | number | 是 | 0 | 路径结束时的y坐标值。 |

****示例：****

1. // xxx.ets
2. @Entry
3. @Component
4. struct BezierCurveTo {
5. private settings: RenderingContextSettings = new RenderingContextSettings(true)
6. private context: CanvasRenderingContext2D = new CanvasRenderingContext2D(this.settings)
7. build() {
8. Flex({ direction: FlexDirection.Column, alignItems: ItemAlign.Center, justifyContent: FlexAlign.Center }) {
9. Canvas(this.context)
10. .width('100%')
11. .height('100%')
12. .backgroundColor('#ffff00')
13. .onReady(() =>{
14. this.context.beginPath()
15. this.context.moveTo(10, 10)
16. this.context.bezierCurveTo(20, 100, 200, 100, 200, 20)
17. this.context.stroke()
18. })
19. }
20. .width('100%')
21. .height('100%')
22. }
23. }



### **HarmonyOS/OpenHarmony应用开发-ArkTS画布组件CanvasRenderingContext2D对象（十五）quadraticCurveTo arc**

### **quadraticCurveTo**

quadraticCurveTo(cpx: number, cpy: number, x: number, y: number): void

创建二次贝赛尔曲线的路径。

****参数：****

| **参数** | **类型** | **必填** | **默认值** | **描述** |
| --- | --- | --- | --- | --- |
| cpx | number | 是 | 0 | 贝塞尔参数的x坐标值。 |
| cpy | number | 是 | 0 | 贝塞尔参数的y坐标值。 |
| x | number | 是 | 0 | 路径结束时的x坐标值。 |
| y | number | 是 | 0 | 路径结束时的y坐标值。 |

****示例：****

1. // xxx.ets
2. @Entry
3. @Component
4. struct QuadraticCurveTo {
5. private settings: RenderingContextSettings = new RenderingContextSettings(true)
6. private context: CanvasRenderingContext2D = new CanvasRenderingContext2D(this.settings)
7. build() {
8. Flex({ direction: FlexDirection.Column, alignItems: ItemAlign.Center, justifyContent: FlexAlign.Center }) {
9. Canvas(this.context)
10. .width('100%')
11. .height('100%')
12. .backgroundColor('#ffff00')
13. .onReady(() =>{
14. this.context.beginPath()
15. this.context.moveTo(20, 20)
16. this.context.quadraticCurveTo(100, 100, 200, 20)
17. this.context.stroke()
18. })
19. }
20. .width('100%')
21. .height('100%')
22. }
23. }



### **arc**

arc(x: number, y: number, radius: number, startAngle: number, endAngle: number, counterclockwise?: boolean): void

绘制弧线路径。

****参数：****

| **参数** | **类型** | **必填** | **默认值** | **描述** |
| --- | --- | --- | --- | --- |
| x | number | 是 | 0 | 弧线圆心的x坐标值。 |
| y | number | 是 | 0 | 弧线圆心的y坐标值。 |
| radius | number | 是 | 0 | 弧线的圆半径。 |
| startAngle | number | 是 | 0 | 弧线的起始弧度。 |
| endAngle | number | 是 | 0 | 弧线的终止弧度。 |
| counterclockwise | boolean | 否 | false | 是否逆时针绘制圆弧。 |

****示例：****

1. // xxx.ets
2. @Entry
3. @Component
4. struct Arc {
5. private settings: RenderingContextSettings = new RenderingContextSettings(true)
6. private context: CanvasRenderingContext2D = new CanvasRenderingContext2D(this.settings)
7. build() {
8. Flex({ direction: FlexDirection.Column, alignItems: ItemAlign.Center, justifyContent: FlexAlign.Center }) {
9. Canvas(this.context)
10. .width('100%')
11. .height('100%')
12. .backgroundColor('#ffff00')
13. .onReady(() =>{
14. this.context.beginPath()
15. this.context.arc(100, 75, 50, 0, 6.28)
16. this.context.stroke()
17. })
18. }
19. .width('100%')
20. .height('100%')
21. }
22. }



### **HarmonyOS/OpenHarmony应用开发-ArkTS画布组件CanvasRenderingContext2D对象（十六）arcTo ellipse**

### **arcTo**

arcTo(x1: number, y1: number, x2: number, y2: number, radius: number): void

依据圆弧经过的点和圆弧半径创建圆弧路径。

****参数：****

| **参数** | **类型** | **必填** | **默认值** | **描述** |
| --- | --- | --- | --- | --- |
| x1 | number | 是 | 0 | 圆弧经过的第一个点的x坐标值。 |
| y1 | number | 是 | 0 | 圆弧经过的第一个点的y坐标值。 |
| x2 | number | 是 | 0 | 圆弧经过的第二个点的x坐标值。 |
| y2 | number | 是 | 0 | 圆弧经过的第二个点的y坐标值。 |
| radius | number | 是 | 0 | 圆弧的圆半径值。 |

****示例：****

1. // xxx.ets
2. @Entry
3. @Component
4. struct ArcTo {
5. private settings: RenderingContextSettings = new RenderingContextSettings(true)
6. private context: CanvasRenderingContext2D = new CanvasRenderingContext2D(this.settings)
7. build() {
8. Flex({ direction: FlexDirection.Column, alignItems: ItemAlign.Center, justifyContent: FlexAlign.Center }) {
9. Canvas(this.context)
10. .width('100%')
11. .height('100%')
12. .backgroundColor('#ffff00')
13. .onReady(() =>{
14. this.context.moveTo(100, 20)
15. this.context.arcTo(150, 20, 150, 70, 50)
16. this.context.stroke()
17. })
18. }
19. .width('100%')
20. .height('100%')
21. }
22. }



### **ellipse**

ellipse(x: number, y: number, radiusX: number, radiusY: number, rotation: number, startAngle: number, endAngle: number, counterclockwise?: boolean): void

在规定的矩形区域绘制一个椭圆。

****参数：****

| **参数** | **类型** | **必填** | **默认值** | **说明** |
| --- | --- | --- | --- | --- |
| x | number | 是 | 0 | 椭圆圆心的x轴坐标。 |
| y | number | 是 | 0 | 椭圆圆心的y轴坐标。 |
| radiusX | number | 是 | 0 | 椭圆x轴的半径长度。 |
| radiusY | number | 是 | 0 | 椭圆y轴的半径长度。 |
| rotation | number | 是 | 0 | 椭圆的旋转角度，单位为弧度。 |
| startAngle | number | 是 | 0 | 椭圆绘制的起始点角度，以弧度表示。 |
| endAngle | number | 是 | 0 | 椭圆绘制的结束点角度，以弧度表示。 |
| counterclockwise | boolean | 否 | false | 是否以逆时针方向绘制椭圆。true:逆时针方向绘制椭圆。false:顺时针方向绘制椭圆。 |

****示例：****

1. // xxx.ets
2. @Entry
3. @Component
4. struct CanvasExample {
5. private settings: RenderingContextSettings = new RenderingContextSettings(true)
6. private context: CanvasRenderingContext2D = new CanvasRenderingContext2D(this.settings)
7. build() {
8. Flex({ direction: FlexDirection.Column, alignItems: ItemAlign.Center, justifyContent: FlexAlign.Center }) {
9. Canvas(this.context)
10. .width('100%')
11. .height('100%')
12. .backgroundColor('#ffff00')
13. .onReady(() =>{
14. this.context.beginPath()
15. this.context.ellipse(200, 200, 50, 100, Math.PI \* 0.25, Math.PI \* 0.5, Math.PI \* 2)
16. this.context.stroke()
17. })
18. }
19. .width('100%')
20. .height('100%')
21. }
22. }



## HarmonyOS/OpenHarmony应用开发-ArkTS画布组件CanvasRenderingContext2D对象（十七）

### **rect**

rect(x: number, y: number, w: number, h: number): void

创建矩形路径。

****参数：****

| **参数** | **类型** | **必填** | **默认值** | **描述** |
| --- | --- | --- | --- | --- |
| x | number | 是 | 0 | 指定矩形的左上角x坐标值。 |
| y | number | 是 | 0 | 指定矩形的左上角y坐标值。 |
| w | number | 是 | 0 | 指定矩形的宽度。 |
| h | number | 是 | 0 | 指定矩形的高度。 |

****示例：****

1. // xxx.ets
2. @Entry
3. @Component
4. struct CanvasExample {
5. private settings: RenderingContextSettings = new RenderingContextSettings(true)
6. private context: CanvasRenderingContext2D = new CanvasRenderingContext2D(this.settings)
7. build() {
8. Flex({ direction: FlexDirection.Column, alignItems: ItemAlign.Center, justifyContent: FlexAlign.Center }) {
9. Canvas(this.context)
10. .width('100%')
11. .height('100%')
12. .backgroundColor('#ffff00')
13. .onReady(() =>{
14. this.context.rect(20, 20, 100, 100) // Create a 100\*100 rectangle at (20, 20)
15. this.context.stroke()
16. })
17. }
18. .width('100%')
19. .height('100%')
20. }
21. }



### **fill**

fill(fillRule?: CanvasFillRule): void

对封闭路径进行填充。

****参数:****

| **参数** | **类型** | **必填** | **默认值** | **描述** |
| --- | --- | --- | --- | --- |
| fillRule | CanvasFillRule | 否 | "nonzero" | 指定要填充对象的规则。可选参数为："nonzero", "evenodd"。 |

****示例:****

1. // xxx.ets
2. @Entry
3. @Component
4. struct Fill {
5. private settings: RenderingContextSettings = new RenderingContextSettings(true)
6. private context: CanvasRenderingContext2D = new CanvasRenderingContext2D(this.settings)
7. build() {
8. Flex({ direction: FlexDirection.Column, alignItems: ItemAlign.Center, justifyContent: FlexAlign.Center }) {
9. Canvas(this.context)
10. .width('100%')
11. .height('100%')
12. .backgroundColor('#ffff00')
13. .onReady(() =>{
14. this.context.rect(20, 20, 100, 100) // Create a 100\*100 rectangle at (20, 20)
15. this.context.fill()
16. })
17. }
18. .width('100%')
19. .height('100%')
20. }
21. }

