

# canvas

canvas简介

canvas基础操作

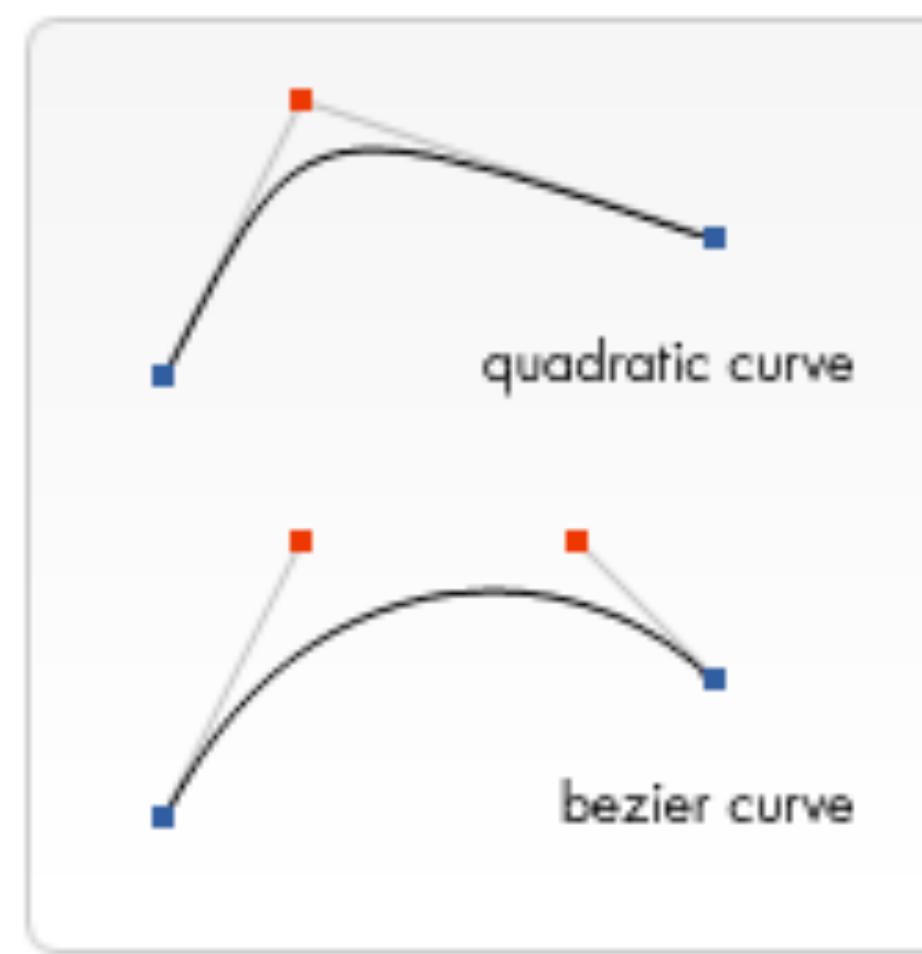
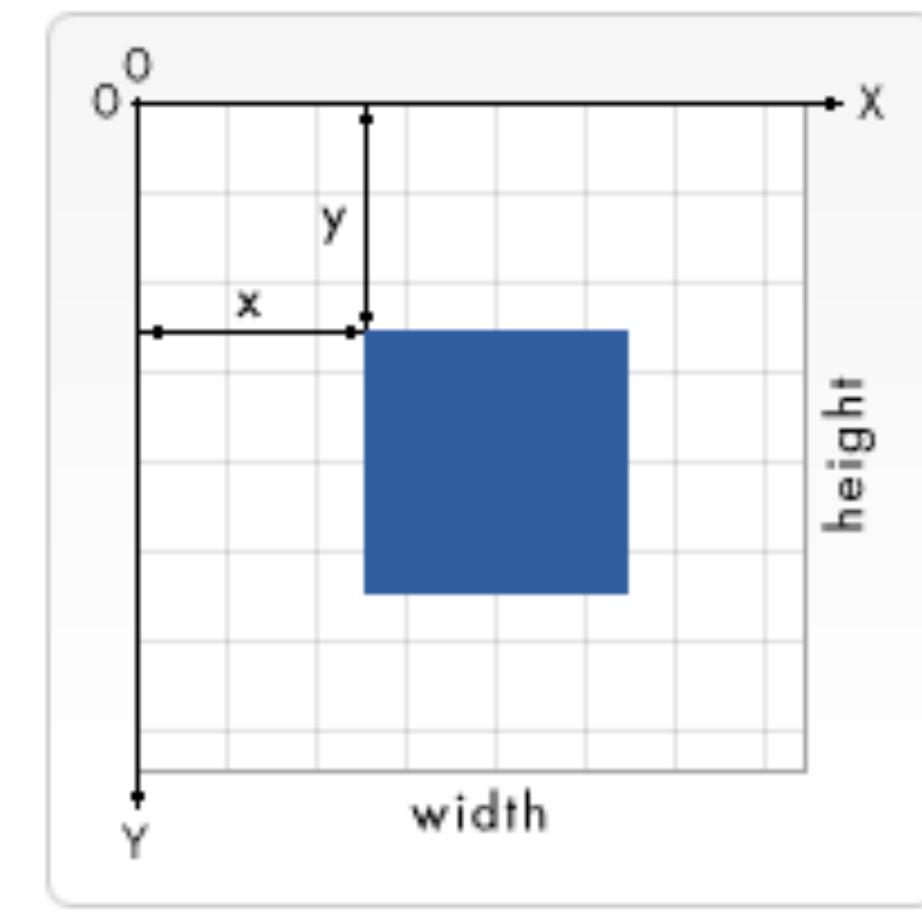
canvas高级操作

# canvas简介

- canvas是H5新增的一个可以使用JS在其中绘制图像的HTML元素，可以用来制作图表、动画，甚至是视频、游戏等。
- canvas最早由Apple引入WebKit，用于Mac OS X的Dashboard和Safari浏览器使用，随后被各大浏览器实现，如今所有主流浏览器都支持该技术。
- canvas属于内联块标签，可以通过css(width、height)设置标签在文档流中的宽高，但是画布的实际宽高仍然是默认值(width: 300px; height: 150px)，如果css样式和画布大小不一致会导致内容扭曲、失真，所以不建议使用css样式来设置，需要通过canvas标签属性(width、height)来设置画布的大小。
- 不支持canvas的浏览器并不会渲染canvas的内容，而是会将canvas标签内的内容渲染出来。

# 使用canvas绘制图形

- `context.fillRect(x, y, w, h)`: 绘制填充矩形
- `context.strokeRect(x, y, w, h)`: 绘制描边矩形
- `context.clearRect(x, y, w, h)`: 清除指定矩形区域，会保留原来的样式
- `context.beginPath()`: 新建一条路径，生成之后，图形绘制命令被指向到路径上生成路径
- `context.closePath()`: 闭合路径之后图形绘制命令又重新指向到上下文中
- `context.stroke([path])`: 通过线条来绘制图形轮廓
- `context.fill([path], [fillRule])`: 通过填充路径的内容区域生成实心的图形
- `context.moveTo(x, y)`: 设置路径起点，将笔触移动到指定的坐标 `x` 以及 `y` 上
- `context.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.rect(x, y, w, h)`: 绘制矩形路径
- `context.roundRect(x, y, w, h, radii)`: 绘制圆角矩形路径
- `context.arc(x, y, radius, startAngle, endAngle, anticlockwise)`: 绘制圆弧/圆
- `context.arcTo(x1, y1, x2, y2, radius)`: 创建介于两个切线之间的弧/曲线
- `context.ellipse(x, y, radiusX, radiusY, rotation, startAngle, endAngle, anticlockwise)`: 绘制椭圆
- `context.quadraticCurveTo(cp1x, cp1y, x, y)`: 二次贝塞尔曲线
- `context.bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)`: 三次贝塞尔曲线
- `context.isPointInPath([path], x, y, [fillRule])`: 判断在当前路径的区域内是否包含检测点
- `context.isPointInStroke([path], x, y)`: 判断在当前路径的描边中是否包含监测点
- `.context.drawFocusIfNeeded([path], element)`: 路径聚焦
- `context.scrollPathIntoView([path])`: 将当前或给定的路径滚动到视口中
- Path2D对象



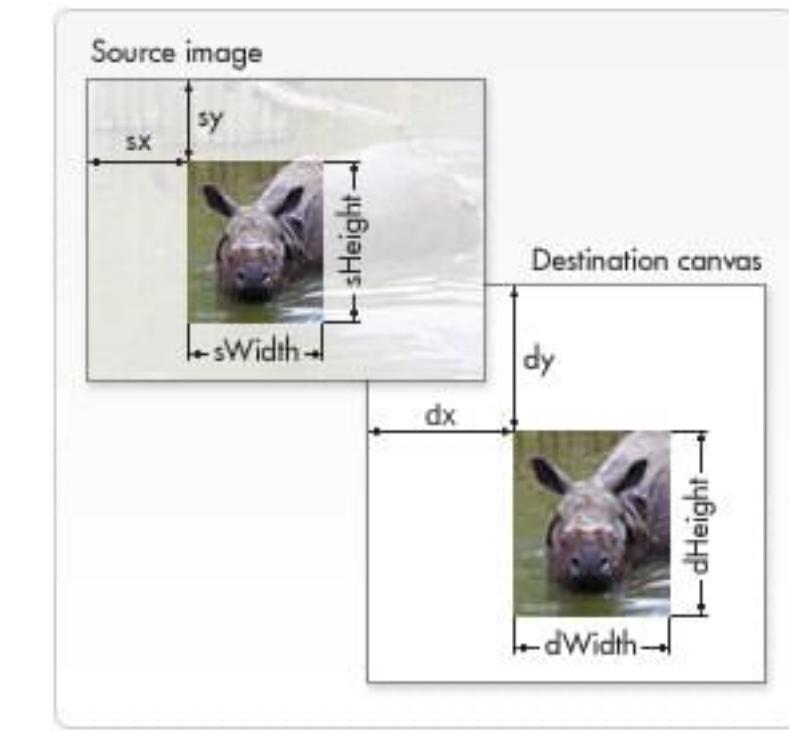
# 应用样式和颜色

- `context.fillStyle = color | gradient | pattern`: 填充颜色
- `context.strokeStyle = color | gradient | pattern`: 轮廓颜色
- `context.globalAlpha = value`: 透明度, 默认 1.0, 取值范围 0.0~1.0
- `context.lineWidth = number`: 线宽
- `context.lineCap = type`: 线帽, 默认 butt, 可选 butt、round、square
- `context.lineJoin = type`: 线条连结方式, 默认 miter, 可选 round、bevel、miter
- `context.miterLimit = number`: 限制两条线相交时交界处内外角顶点的长度, 默认 10
- `context.shadowColor = color`: 阴影颜色默认 `rgba(0, 0, 0, 0)`
- `context.shadowBlur = level`: 模糊度, 默认 0
- `context.shadowOffsetX = offset`: 阴影水平偏移, 默认 0
- `context.shadowOffsetY = offset`: 阴影垂直偏移, 默认 0
- `context.getLineDash()`: 返回一个包含当前虚线样式, 长度为非负偶数的数组
- `context.setLineDash(segments)`: 设置虚线样式
  - `segments`: 一组描述交替绘制线段和间距长度的数字, 如果数组元素的数量是奇数, 数组的元素会被复制并重复
- `context.lineDashOffset = float`: 虚线的起始偏移量, 默认 0.0
- `context.createLinearGradient(x0, y0, x1, y1)`: 线型渐变, 返回`CanvasGradient`对象
  - `CanvasGradient.addColorStop(offset, color)`: 添加断点到渐变中
- `context.createRadialGradient(x0, y0, r0, x1, y1, r1)`: 径向渐变, 返回`CanvasGradient`对象
  - `CanvasGradient.addColorStop(offset, color)`: 添加断点到渐变中
- `context.createConicGradient(startAngle, x, y)`: 锥形渐变, 返回`CanvasGradient`对象
  - `CanvasGradient.addColorStop(offset, color)`: 添加断点到渐变中
- `context.createPattern(image, repetition)`: 图案样式, 返回`CanvasPattern`对象
  - `CanvasPattern.setTransform(matrix)`: 创建一个来自`DOMMatrix`具有指定图案变化的`CanvasPattern`

# 绘制文本

- `context.fillText(text, x, y, [maxWidth])`
- `context.strokeText(text, x, y, [maxWidth])`
- `context.font = value`: 字体样式, 同CSS `font`语法相同, 默认 `10px sans-serif`
- `context.textAlign = value`: 对齐方式, 默认 `start`, 可选 `start`、`end`、`left`、`right`、`center`
- `context.textBaseline = value`: 基线对齐方式, 默认 `alphabetic`, 可选 `top`、`hanging`、`middle`、`alphabetic`、`ideographic`、`bottom`
- `context.direction = value`: 文本方向, 默认 `inherit`, 可选 `ltr`、`rtl`、`inherit`
- `context.fontKerning = value`: 指定如何使用字体字距调整信息, 可选 `auto`、`normal`、`none`
- `context.fontStretch = value`: 指定绘制文本时如何扩展或压缩字体, 同CSS `font-stretch`语法相同
- `context.letterSpacing = value`: 字符间距, 默认 `0px`
- `context.wordSpacing = value`: 单词间距, 默认 `0px`
- `context.fontVariantCaps = value`: 指定呈现文本的替代大小写, 默认 `normal`, 可选 `normal`、`small-caps`、`all-small-caps`、`petite-caps`、`all-petite-caps`、`unicase`、`titling-caps`
- `context.textRendering = value`: 文本渲染的优化方式, 同CSS `text-rendering`语法相同
- `context.measureText(text)`: 测量文本, 返回被测量文本TextMetrics对象

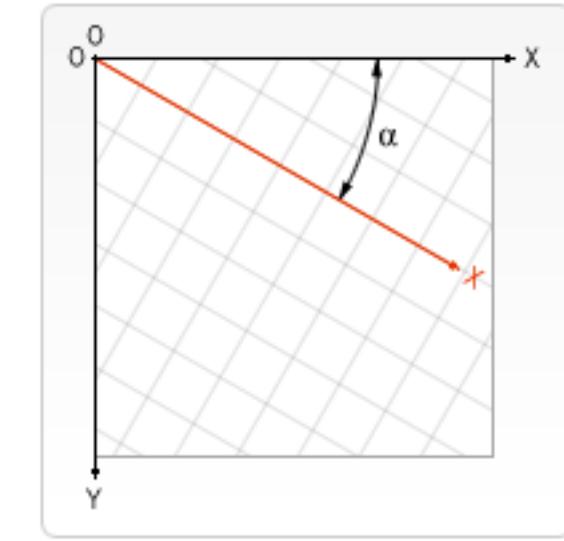
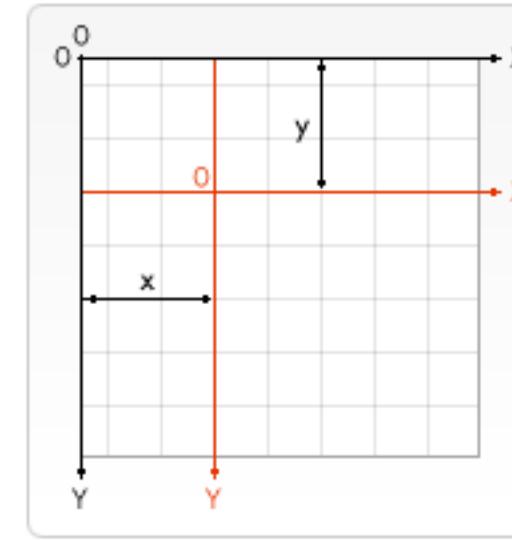
# 使用图像



- `context.drawImage`
  - `context.drawImage(image, dx, dy)`: 将图片绘制到画布指定位置
  - `context.drawImage(image, dx, dy, dw, dh)`: 将图片绘制到画布的指定区域
  - `context.drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)`: 截取图片的指定区域并绘制到画布的指定区域
- `context.imageSmoothingEnabled = bool`: 设置图片缩放时是否平滑, 默认 `true`
- `context.imageSmoothingQuality = value`: 设置图片平滑质量, 默认 `low`, 可选 `low`、`medium`、`high` (前提: `context.imageSmoothingEnabled = true`)

# 变换

- `context.save()`: 保存画布的状态
- `context.restore()`: 恢复画布的状态
- `context.reset()`: 重置画布为默认状态，允许将其重用于绘制其他内容，而无需显式重置所有属性
- `context.translate(x, y)`: 平移
- `context.scale(x, y)`: 缩放
- `context.rotate(angle)`: 旋转
- `context.transform(a, b, c, d, e, f)`: 矩阵变换
- `context.getTransform()`: 获取当前被应用到上下文的矩阵
- `context.setTransform(a, b, c, d, e, f)`: 矩阵变换
- `context.resetTransform()`: 重置为单位矩阵，和调用`context.setTransform(1, 0, 0, 1, 0, 0)`效果一样



# 组合

- `context.globalCompositeOperation = type`

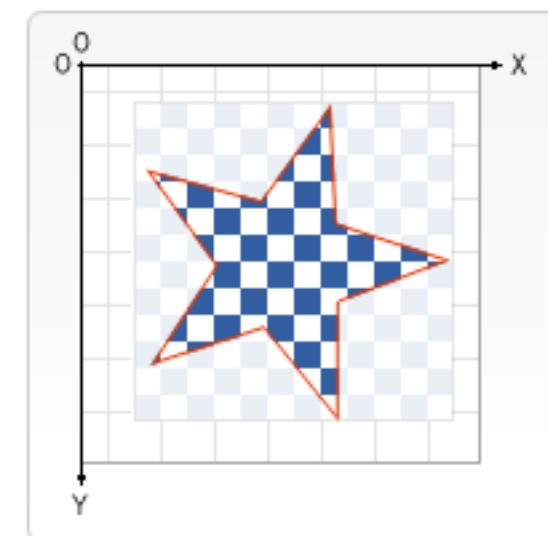
源 (source) : 要绘制的新图形

目标 (destination) : 已经在页面上绘制好的图像

- `source-over`: 在目标图像上显示源图像 (默认值)
- `destination-over`: 在源图像上方显示目标图像
- `source-atop`: 在目标图像顶部显示源图像, 源图像位于目标图像之外的部分是不可见的
- `destination-atop`: 在源图像顶部显示目标图像, 目标图像位于源图像之外的部分是不可见的
- `source-in`: 绘制重合部分的源
- `destination-in`: 绘制重合部分的目标
- `source-out`: 绘制未重合部分的源
- `destination-out`: 绘制未重合部分的目标
- `lighter`: 显示源图像 + 目标图像(融合)
- `copy`: 显示源图像, 忽略目标图像
- `xor`: 绘制未重合部分的 源 和 目标

- `context.clip`

- `context.clip()`: 将当前正在构建的路径转换为当前的裁剪路径
- `context.clip(fillRule)`: 将当前正在构建的路径转换为当前的裁剪路径
  - `fillRule`: 裁剪规则, 默认 `nonzero`, 可选 `nonzero`、`evenodd`
- `context.clip(path, fillRule)`: 将当前正在构建的路径转换为当前的裁剪路径
  - `fillRule`: 裁剪规则, 默认 `nonzero`, 可选 `nonzero`、`evenodd`
  - `path`: 需要剪切的Path2D路径



# 像素操作

- context.createImageData
  - context.createImageData(sw, sh): 创建一个新的具体特定尺寸的 ImageData 对象。所有像素被预设为透明黑色 (rgba(0, 0, 0, 0))
  - context.createImageData(imageData): 创建一个被 anotherImageData 对象指定的相同像素的 ImageData 对象。这个新的对象像素全部被预设为透明黑色 (rgba(0, 0, 0, 0))。这个并非复制了图片数据
- context.getImageData(sx, sy, sw, sh): 获取canvas指定区域的像素点，返回 ImageData 对象，对象中的 data、width、height 属性
- context.putImageData
  - context.putImageData(imageData, dx, dy): 在 canvas 指定位置绘制像素
  - context.putImageData(imageData, dx, dy, dirtyX, dirtyY, dirtyWidth, dirtyHeight): 在 canvas 指定位置绘制像素

# canvas的实例属性

- `canvas.height = number`: 画布的高度，当该属性没有被定义，或被定义为一个无效值（如负值）时，将使用150作为它的默认值，支持读写。
- `canvas.width = number`: 画布的宽度，当这个属性没有被定义，或被定义为一个无效值（如负值）时，将使用300作为它的默认值，支持读写。
- `canvas.mozOpaque = bool`: （非标准，不建议使用）控制画布是否半透明，支持读写。

# canvas的实例方法

- `canvas.captureStream([frameRate])`: 返回`CanvasCaptureMediaStream`对象，一个实时视频捕获的画布
  - `frameRate`: 设置双精准度浮点值为每个帧的捕获速率。如果未设置，则每次画布更改时都会捕获一个新帧。如果设置为0，则会捕获单个帧
- `canvas.getContext(contextType, [contextAttributes])`: 返回`canvas`的上下文对象，如果上下文没有定义则返回`null`
  - `contextType`: 上下文类型
    - `2d`: 创建一个`CanvasRenderingContext2D`二维渲染上下文对象
    - `webgl` (或`experimental-webgl`) : 创建一个`WebGLRenderingContext`三维渲染上下文对象
    - `webgl2` (或`experimental-webgl2`) : 创建一个`WebGL2RenderingContext`三维渲染上下文对象
    - `bitmaprederer`: 创建一个`ImageBitmapRenderingContext`上下文对象，只提供将`canvas`内容替换为指定`ImageBitmap`功能
  - `contextAttributes`: 上下文属性
- `canvas.toDataURL([type], [quality])`: 返回一个包含图片展示的data URI
  - `type`: 图片格式，默认 `image/png`
  - `quality`: 图片质量，默认 `0.92`，取值范围 `0~1`
- `canvas.toBlob(callback, [type], [quality])`: 创建一个Blob对象
  - `callback`: 回调函数，可获得一个单独的Blob对象参数，如果图像未被成功创建，可能会获得`null`值
  - `type`: 图片格式，默认 `image/png`
  - `quality`: 图片质量，默认 `0.92`，取值范围 `0~1`
- `canvas.transferControlToOffscreen()`: 创建一个`OffscreenCanvas`对象，一个可以脱离屏幕渲染的`canvas`对象，它在窗口环境和web worker环境均有效

# canvas的事件

- `canvas.oncontextlost()`: 浏览器检测到CanvasRenderingContext2D上下文环境丢失时触发
- `canvas.oncontextrestored()`: 浏览器还原CanvasRenderingContext2D上下文环境时触发
- `canvas.webglcontextcreationerror()`: 浏览器无法创建WebGLRenderingContext上下文环境时触发
- `canvas.onwebglcontextlost()`: 浏览器检测到WebGLRenderingContext上下文环境丢失时触发
- `canvas.onwebglcontextrestored()`: 浏览器还原WebGLRenderingContext上下文环境时触发

# 性能优化

- 在离屏canvas上预渲染相似或重复的对象
- 避免使用浮点数坐标
- 不要在用drawImage时缩放图像
- 使用多层画布去绘制一个复杂的场景
- 使用CSS设置大的背景图
- 使用CSS变换特性缩放画布
- 关闭透明度
- 将画布的函数调用集合到一起
- 避免不必要的画布状态改变
- 渲染画布中的不同的点，而非更新状态
- 尽可能避免shadowBlur特性
- 尽可能避免textRendering
- 尝试不同的方法来清除画布
- 绘制动画时尽可能用window.requestAnimationFrame()替代window.setInterval()