第5章

微信小程序canvas画布组件应用

微信小程序提供了 canvas 画布组件，可以在页面中定义一个画布，然后使用 canvas 绘图方法在画布中进行画任何的线、图形、填充等一系列的操作。在游戏开发中会大量使用 canvas 画图。本章介绍在微信小程序中如何使用 canvas 进行画图操作以及实现游戏动画。

5.1 canvas 画布组件

5.1.1 画布 canvas

canvas 为画布组件，其默认尺寸是宽度 300px、高度 225px。该组件的属性如表 5-1 所示。

表 5-1 canvas 组件的属性

属性名	类型	默认值	说 明
canvas-id	String		canvas 组件的唯一标识符
disable-scroll	Boolean	false	当在 canvas 中移动且有绑定手势事件时，禁止屏幕滚动以及下拉刷新
bindtouchstart	EventHandle		手指触摸动作开始
bindtouchmove	EventHandle		手指触摸后移动
bindtouchend	EventHandle		手指触摸动作结束
bindtouchcancel	EventHandle		手指触摸动作被打断，如来电提醒、弹窗
bindlongtap	EventHandle		手指长按 500ms 之后触发，触发了长按事件后进行移动不会触发屏幕的滚动

例如定义一个画布 WXML 代码如下：

```
<canvas canvas-id="myCanvas" style="width: 300px; height: 300px;" ></canvas>
```

注意：canvas 标签默认宽度 300px、高度 225px。在同一页面中的 canvas-id 不可重复，如果使用一个已经出现过的 canvas-id，该 canvas 标签对应的画布将被隐藏并不再正常工作。

示例代码：

```
<!-- canvas.wxml -->
<canvas style="width: 300px; height: 200px;" canvas-id="firstCanvas"></canvas>
<!-- 当使用绝对定位时，文档流后边的 canvas 的显示层级高于前边的 canvas -->
<canvas style="width: 400px; height: 500px;" canvas-id="secondCanvas"></canvas>
```

```
// canvas.js
Page({
  canvasIdErrorCallback: function (e) {
    console.error(e.detail.errMsg)
  },
  onReady: function (e) {
    //使用 wx.createContext 获取绘图上下文 context
    var context = wx.createCanvasContext('firstCanvas')
    context.setStrokeStyle("#00ff00")
    context.setLineWidth(5)
    context.rect(0,0,200,200)
    context.stroke()
    context.setStrokeStyle ("#ff0000")
    context.setLineWidth(2)
    context.moveTo(160,100)
    context.arc(100,100,60,0,2 * Math.PI,true)
    context.moveTo(140,100)
    context.arc(100,100,40,0,Math.PI,false)
    context.moveTo(85,80)
    context.arc(80,80,5,0,2 * Math.PI,true)
    context.moveTo(125,80)
    context.arc(120,80,5,0,2 * Math.PI,true)
    context.stroke()
    context.draw()
  }
})
```

5.1.2　响应 canvas 组件事件

canvas 组件可以响应手指触摸动作。可以在< canvas >中加上一些事件，来观测手指的坐标。

【例 5-1】　观测手指触摸的坐标。

WXML 代码如下：

```
//index.wxml
<canvas canvas - id = "myCanvas" style = "margin: 5px; border:1px solid #d3d3d3;"
  bindtouchstart = "start" bindtouchmove = "move" bindtouchend = "end"/>
<view hidden = "{{hidden}}">
  Coordinates: ({{x}}, {{y}})
</view>
```

其中“canvas-id”为当前画布的名称，“bindtouchstart”是单击后才会触发的，“bindtouchend”是手指触摸动作结束才会触发的，“bindtouchmove”是手指触摸后移动时便会触发并且可以传过来目前移动的参数坐标。例如：

```
move: function( event ) {
        var xx = event.touches[0].x;
        var yy = event.touches[0].y;
        console.log(xx + "," + yy)
    },
```

这里代码实现手指触摸后移动时打印坐标。

Index.js 文件完整代码如下：

```
Page({
  data: {
    x: 0, y: 0,
    hidden: true
  },
  start: function(e) {
    this.setData({
      hidden: false,
      x: e.touches[0].x,
      y: e.touches[0].y
    })
  },
  move: function(e) {
    this.setData({
      x: e.touches[0].x,
      y: e.touches[0].y
    })
  },
  end: function(e) {
    this.setData({
      hidden: true
    })
  }
})
```

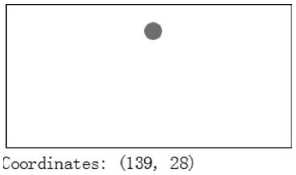


图 5-1　显示出手指触碰点的坐标

当把手指放到 canvas 中移动，就会在下边显示出触碰点的坐标，如图 5-1 所示。

在游戏开发中往往需要根据手指触摸单击动作下棋、移动物体等，都是利用这些 bindtouchstart、bindtouchmove 和 bindtouchend 事件来实现的。

5.2　使用 canvas 画图

小程序中 canvas 画布组件可以在页面中定义一个画布，然后使用画布上下文对象的绘图方法在画布中进行画任何的线、图形、填充等一系列的操作。本节介绍小程序中如何使用 canvas 画图，为游戏开发打下基础。

5.2.1　canvas 组件定义语法

canvas 组件的定义语法如下：

```
< canvas canvas - id = "xxx" bindtouchstart = "xxxx" > … </canvas >
```

canvas-id 是 canvas 组件的标识；bindtouchstart 是手指触摸动作开始事件。

例如在小程序中定义一个 canvas 画布，id 为 myCanvas，高和宽各为 300 像素，WXML

代码如下：

```
<canvas canvas - id = "myCanvas" style = "width: 300px; height: 300px;" ></canvas >
```

要在 canvas 组件中绘图还需要获得 canvas 组件的上下文对象，可以在 JS 页面的 onLoad 函数中使用 API 绘图接口 wx.createCanvasContext()创建画布上下文对象，代码如下：

```
//使用 wx.createContext()创建绘图上下文 context
var context = wx.createCanvasContext('myCanvas')
```

canvas 绘制图形都是依靠 canvas 组件的上下文对象。上下文对象用于定义如何在画布上绘图，上下文对象支持在画布上绘制 2D 图形、图像和文本。

5.2.2　坐标系统

在实际的绘图中，我们所关注的一般都是指设备坐标系，此坐标系以像素为单位，像素指的是屏幕上的亮点。每个像素都有一个坐标点与之对应，左上角的坐标设为(0,0)，向右为 X 正轴，向下为 Y 正轴。一般情况下以(x,y)代表屏幕上某个像素的坐标点，其中水平以 X 坐标值表示，垂直以 Y 坐标值表示。例如，在图 5-2 所示的坐标系中画一个点，该点的坐标(x, y)是(4, 3)。

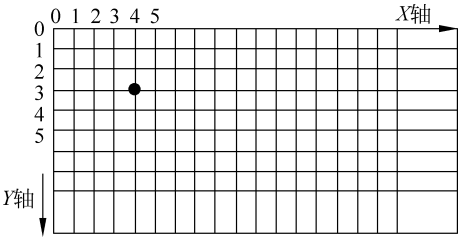


图 5-2　canvas 坐标的示意图

小程序作图是在一个事先定义好的坐标系中进行的，这与日常生活中的绘图方式有着很大的区别。图形的大小、位置等都与绘图区或容器的坐标有关。

5.2.3　颜色的表示方法

1. 颜色关键字

支持 148 种颜色名，它们是 aqua、black、blue、fuchsia、gray、green、lime、maroon、navy、olive、purple、red、silver、teal、white、yellow 等。如果需要使用其他的颜色，需要使用十六进制的颜色值。

例如：

```
context.setStrokeStyle('green')      #设置线条样式为绿色
```

2. 十六进制颜色值

可以使用一个十六进制字符串表示颜色，格式为＃RGB。其中，R 表示红色分量，G 表示绿色分量，B 表示蓝色分量。每种颜色的最小值是 0(十六进制：＃00)。最大值是 255(十六进制：＃FF)。例如＃FF0000 表示红色，＃00FF00 表示绿色，＃0000FF 表示蓝色，＃A020F0 表示紫色，＃FFFFFF 表示白色，＃000000 表示黑色。

例如：

```
context.setStrokeStyle('＃00FF00')      #设置线条样式为绿色
```

3. RGB 颜色值

RGB 颜色值可以使用如 rgb(红色分量，绿色分量，蓝色分量)形式表示颜色，分量范围为 0 到 255 的整数。表 5-2 是十六进制字符串表示颜色与 RGB 颜色值对照表。

表 5-2 十六进制字符串表示颜色与 RGB 颜色值对照表

Color HEX	Color RGB	颜色	Color HEX	Color RGB	颜色
#000000	rgb(0,0,0)	黑色	#00FFFF	rgb(0,255,255)	青色
#FF0000	rgb(255,0,0)	红色	#FF00FF	rgb(255,0,255)	深红
#00FF00	rgb(0,255,0)	绿色	#C0C0C0	rgb(192,192,192)	灰色
#0000FF	rgb(0,0,255)	蓝色	#FFFFFF	rgb(255,255,255)	白色
#FFFF00	rgb(255,255,0)	黄色	#FF8000	rgb(255,128,0)	橘黄

例如：

```
context.setStrokeStyle('rgb(0, 255, 0)') #设置线条样式为绿色
```

4. RGBA 颜色

与 RGB 颜色值类似，前三个值是红绿蓝分量，第四个是 alpha 值，指定色彩的透明度，它的范围为 0.0 到 1.0，0.5 为半透明。例如：rgba(255,255,255,0)则表示完全透明的白色；rgba(0,0,0,1)则表示完全不透明的黑色；rgba(0,0,0,0)则表示完全透明的黑色，也即是无色。

5.2.4 绘制直线

在小程序中绘制直线，具体过程如下：

(1) 获取 canvas 组件的上下文对象 context。

```
const context = wx.createCanvasContext('myCanvas')        //获取 canvas 的上下文对象
```

(2) 调用 beginPath()方法，指示开始绘图路径，即开始绘图。语法如下：

```
context.beginPath();
```

(3) 调用 moveTo()方法将坐标移至直线起点。语法如下：

```
context.moveTo(x,y);
```

x 和 y 为要移动至的坐标。

(4) 调用 lineTo()方法绘制直线。语法如下：

```
context.lineTo(x,y);
```

x 和 y 为直线的终点坐标。

(5) 调用 stroke()方法，绘制图形的边界轮廓。语法如下：

```
context. stroke();
```

(6) 调用 draw()方法把全部路径绘制到画布上。语法如下：

```
context. draw();                    //清空画布原来绘画再继续绘制
```

draw()方法将之前在绘图上下文中的描述(路径、变形、样式)画到 canvas 中。其中参数是 Boolean 型,未写参数时默认是 false,表示清空画布原来绘画再继续绘制。参数是 true 时,表示保留当前画布上内容,本次调用 drawCanvas 绘制的内容覆盖在原有内容上面。

```
context. draw(true);                //保留画布原来绘画继续绘制
```

【例 5-2】 使用连续画线的方法绘制一个三角形。

WXML 代码如下:

```
<canvas canvas - id = "myCanvas" style = "width: 300px; height: 300px;" ></canvas >
```

JS 代码如下:

```
Page({
  onLoad: function (options) {
    this.drawtriangle();
  },
  drawtriangle: function()
  {
    const ctx = wx.createCanvasContext('myCanvas')      //创建画布上下文
    ctx.beginPath();                                    //开始绘图路径
    ctx.moveTo(100, 0);                                 // 将坐标移至直线起点
    ctx.lineTo(50, 100);                                // 绘制直线
    ctx.lineTo(150, 100);                               // 绘制直线
    ctx.lineTo(100, 0);                                 // 绘制直线
    ctx.closePath();              //闭合路径,不是必需的,如果线的终点跟起点一样会自动闭合
    ctx.stroke();                                       // 通过线条绘制轮廓(边框)
    ctx.draw(true);                                     // 全部绘制到画布上
  },
})
```

运行结果如图 5-3 所示。

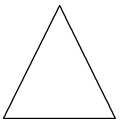


图 5-3　canvas 绘制一个三角形

【例 5-3】 通过画线绘制复杂菊花图形。

WXML 代码如下:

```
<canvas canvas - id = "myCanvas" style = "width: 300px; height: 300px;" >
</canvas >
```

JS 代码如下:

```
Page({
  onLoad: function(options) {
    const ctx = wx.createCanvasContext('myCanvas')      //创建画布上下文
    var dx = 150;
    var dy = 150;
    var s = 100;
    ctx.beginPath();                                    //开始绘图路径
    var x = Math.sin(0);
```

```
    var y = Math.cos(0);
    var dig = Math.PI / 15 * 11;
    for (var i = 0; i < 30; i++) {
      var x = Math.sin(i * dig);
      var y = Math.cos(i * dig);
      //用三角函数计算顶点
      ctx.lineTo(dx + x * s, dy + y * s);
    }
    ctx.closePath();
    ctx.stroke();
    ctx.draw(true);
  },
})
```

运行结果如图 5-4 所示。

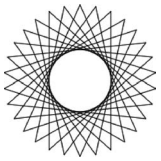


图 5-4　canvas 绘制复杂图形

5.2.5　绘制矩形

可以通过调用 rect()、strokeRect()、fillRect()和 clearRect()4 个方法在 canvas 画布中绘制矩形。其中，前两个方法用于绘制矩形边框，调用 fillRect()可以填充指定的矩形区域，调用 clearRect ()可以擦除指定的矩形区域。

1. 创建矩形

小程序使用画布对象的 rect()方法创建矩形，然后使用 fill()或 stroke()方法在画布上填充实心矩形或描边空心矩形。其语法格式如下：

```
context.rect(x, y, width, height)
```

参数解释如下。

(1) x：Number 类型，矩形左上角点的 x 坐标。

(2) y：Number 类型，矩形左上角点的 y 坐标。

(3) width：Number 类型，矩形的宽度。

(4) height：Number 类型，矩形的高度。

【例 5-4】 绘制宽、高均为 200 像素的矩形。

WXML 代码如下：

```
<canvas canvas - id = "myCanvas" style = "width: 300px; height: 300px;" ></canvas >
```

JS 代码如下：

```
Page({
  onLoad: function (options) {
```

```
    //创建画布上下文
    const context = wx.createCanvasContext('myCanvas')
    //描述一个左上角坐标为(50,50),宽、高均为 200 像素的矩形
    context.rect(50, 50, 200, 200)
    context.setFillStyle('orange')      //描述填充颜色为橙色
    context.fill()                      //描述填充矩形动作
    context.draw()                      //在画布上执行全部描述
  }
})
```

注意画笔默认是黑色效果(无论是填充还是描边)。setFillStyle 用于设置画笔填充颜色,这里仅为临时使用。

上下文 context 对象提供表 5-3 所示的方法来设置样式(颜色)、阴影和线条宽度。

表 5-3　context 对象提供的方法

方　法	说　明
setFillStyle	设置填充样式(颜色),如果没有设置 FillStyle,默认颜色为 black
setStrokeStyle	设置线条样式(颜色),如果没有设置 StrokeStyle,默认颜色为 black
setShadow	设置阴影
setLineWidth	设置线条边线线宽

2. 填充矩形

小程序使用画布对象的 fillRect()方法直接在画布上填充实心矩形,其语法格式如下:

```
context.fillRect(x, y, width, height)
```

参数与创建矩形的 rect()方法参数完全相同。

3. 描边矩形

小程序使用画布对象的 strokeRect()方法直接在画布上描边空心矩形,其语法格式如下:

```
context.strokeRect(x, y, width, height)        //矩形画有线条
```

参数与创建矩形的 rect()方法参数完全相同。

4. 清空矩形区域

小程序使用画布对象的 clearRect()方法清空矩形区域。其语法格式如下:

```
context. clearRect(x, y, width, height)     //清空矩形区域
```

参数与创建矩形的 rect()方法参数完全相同。

5.2.6　绘制圆弧

可以调用 arc()方法绘制圆弧,语法如下:

```
arc(centerX, centerY, radius, startingAngle, endingAngle, antiClockwise);
```

参数说明:centerX,圆弧圆心的 X 坐标;centerY,圆弧圆心的 Y 坐标;radius,圆弧的

半径；startingAngle，圆弧的起始角度；endingAngle，圆弧的结束角度；antiClockwise，是否按逆时针方向绘图。

例如：使用 arc()方法绘制圆心为(50,50)，半径为 100 的圆弧。圆弧的起始角度为 60 度，圆弧的结束角度为 180 度。

```
context.beginPath();      // 开始绘图路径
context.arc(50, 50, 100, 1/3 * Math.PI, 1 * Math.PI, false);
context.stroke();
```

【例 5-5】 使用 arc()方法画圆。

WXML 代码如下：

```
<canvas canvas - id = "myCanvas" style = "width: 400px; height: 400px;" ></canvas >
```

JS 代码如下：

```
Page({
  onLoad: function (options) {
    this.drawArc ();
  },
  drawArc :function ()
  {
    //创建画布上下文
    const context = wx.createCanvasContext('myCanvas')
     //用循环绘制 10 个圆形
     var n = 0;
     for(var i = 0 ;i < 10;i++){
         //开始创建路径,因为圆本质上也是一个路径,这里向 canvas 说明要开始画了,这是起点
        context.beginPath();
        context.arc(i * 25,i * 25,i * 10,0,Math.PI * 2,true);
        context.fillStyle = "rgba(255,0,0,0.25)";
        context.fill();          //填充刚才所画的圆形
     }
    context.draw()               //在画布上执行全部描述
  },
})
```

图 5-5 canvas 绘制圆

运行结果如图 5-5 所示。

5.2.7 绘制图像

在画布上绘制图像的方法是 drawImage()，语法如下：

```
drawImage(image, x, y)
drawImage(image, x, y, width, height)
drawImage ( image,  sourceX,  sourceY,  sourceWidth,
sourceHeight,destX, destY, destWidth, destHeight)
```

参数说明：

image，所要绘制的图像。

x 和 y，要绘制的图像的左上角位置；width 和

height，绘制图像的宽度和高度。

sourceX 和 sourceY，图像将要被绘制的区域的左上角；destX 和 destY，所要绘制的图像区域的左上角的画布坐标；sourceWidth，sourceHeight 被绘制的原图像区域；destWidth 和 destHeight，图像区域在画布上要绘制成的大小。

【例 5-6】 不同形式显示图书封面 cover.png，其宽度是 240 像素，高度是 320 像素。

WXML 代码如下：

```
<canvas canvas-id="myCanvas" style="width: 400px; height: 600px;" ></canvas>
```

JS 代码如下：

```
Page({
  onLoad: function (options) {
    //创建画布上下文
    const ctx = wx.createCanvasContext('myCanvas')
    var imageObj = "/images/cover.png"          //图片本地路径
    ctx.drawImage(imageObj, 0, 0);              //原图大小显示
    ctx.drawImage(imageObj, 250, 0, 120, 160);  //原图一半大小显示
    //从原图(0,100)位置开始截取中间一块宽 240、高 160 的区域，原大小显示在屏幕(50,350)处
    ctx.drawImage(imageObj, 0, 100, 240, 160, 50, 350, 240, 160);
    ctx.draw();
  },
})
```

运行结果如图 5-6 所示。

图 5-6　不同形式显示图书封面

5.2.8　输出文字

可以使用 strokeText()方法在画布的指定位置输出文字，语法如下：

```
strokeText(string text, x, y, maxWidth)
```

参数说明：string 为所要输出的字符串；x 和 y 为要输出的字符串位置坐标；maxWidth 是绘制文本的最大宽度，为可选项。

例如：

```
context.strokeText("中原工学院", 100, 100);
//在(100, 100)处显示"中原工学院"
```

1. 设置字体

小程序提供 setFontSize()方法用于设置字体大小，格式如下：

```
context. setFontSize(fontSize)
```

例如：

```
context. setFontSize(40);          //表示字体的字号为 40 号
context.strokeText("中原工学院", 100, 100);
```

小程序提供 font 属性自定义字体风格，其格式如下：

```
context.font = value
```

参数 value 的默认值为 10px sans-serif，表示字体大小为 10px、字体家族为 sans-serif。value 支持的属性如下。

(1) style：字体样式，仅支持 italic、oblique、normal。

(2) weight：字体粗细。仅支持 normal、bold。

(3) size：字体大小。

(4) family：字体族名。注意确认各平台所支持的字体。

例如：

```
context.font = "bold 40px 隶书"
context.strokeText("中原工学院", 100, 100);
```

上述代码表示字体设置为加粗、字体大小为 40 像素、隶书字体样式。

2. 设置对齐方式

可以通过 setTextAlign()来设置输出字符串的对齐方式。可选值为"left"(左对齐)、"center"(居中对齐)和"right"(右对齐)。文本对齐参照效果如图 5-7 所示。

例如：

```
context.setTextAlign("center")
```

或

```
context.textAlign = "center"        //基础库 1.9.90 开始支持
```

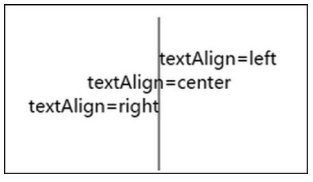


图 5-7　文本对齐参照效果

3. 设置文字颜色

可以通过设置 canvas 的上下文对象的 strokeStyle 属性指定输出文字的颜色。

```
context.strokeStyle = "blue";
context.strokeText("中原工学院", 100, 100);
```

4. 填充字体内部

使用 strokeText()方法输出的文字是中空的，只绘制了边框。如果要填充文字内部，可以使用 fillText()方法，语法如下：

```
fillText (string text, x, y)
```

可以使用 context.fillStyle 属性指定填充的颜色。

```
context.fillStyle = "blue";
```

【例 5-7】 渐变填充文字。

WXML 代码如下：

```
<canvas canvas - id = "myCanvas" style = "width: 400px; height: 400px;" ></canvas >
```

JS 代码如下：

```
Page({
  onLoad: function (options) {
    //创建画布上下文
    const ctx = wx.createCanvasContext('myCanvas')
  var Colordiagonal = ctx.createLinearGradient(100,100, 300,100);
  Colordiagonal.addColorStop(0, "yellow");
  Colordiagonal.addColorStop(0.5, "green");
  Colordiagonal.addColorStop(1, "red");
  ctx.fillStyle = Colordiagonal;
  ctx.font = "30px 隶书";
  ctx.fillText("中原工学院", 100, 100);
  ctx.draw();
  },
})
```

运行结果如图 5-8 所示。

图 5-8　渐变填充文字

5.2.9　保存和恢复绘图状态

在绘制复杂图形时有可能临时需要进行多个属性的设置更改(例如画笔的粗细、填充颜色等效果),在绘制完成后又要重新恢复初始设置进行后续的操作。调用 context. save()方法可以保存当前的绘图状态。绘图状态包括：(1)当前应用的操作(比如移动、旋转、缩放或变形,具体方法将在本节稍后介绍)。(2) strokeStyle、fillStyle、globalAlpha、lineWidth、lineCap、lineJoin、miterLimit、shadowOffsetX、shadowOffsetY、shadowBlur、shadowColor、globalCompositeOperation 等属性的值。

调用 context. restore()方法可以从堆栈中弹出之前保存的绘图状态。

context. save()方法和 context. restore()方法都没有参数。

【例 5-8】 保存和恢复绘图状态。

WXML 代码如下：

```
<canvas canvas - id = "myCanvas" style = "width: 300px; height: 300px;" ></canvas >
```

JS 代码如下：

```
Page({
  onLoad: function(options) {
    //创建画布上下文
    const ctx = wx.createCanvasContext('myCanvas')
    ctx.fillStyle = 'red'
    ctx.fillRect(0, 0, 150, 150);   // 使用红色填充矩形
    ctx.save();                     // 保存当前的绘图状态
    ctx.fillStyle = 'green'//
    ctx.fillRect(45, 45, 60, 60);   //使用绿色填充矩形
    ctx.restore();                  // 恢复在之前保存的绘图状态,即 ctx.fillStyle = 'red'
    ctx.fillRect(60, 60, 30, 30);   // 使用红色填充矩形
    ctx.draw();
  },
})
```

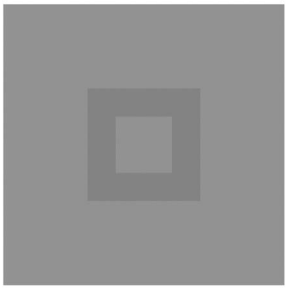


图 5-9　保存和恢复绘图状态

运行结果如图 5-9 所示。

5.2.10　图形的变换

1. 平移 translate(x,y)

移动图形到新的位置,图形的大小、形状不变。参数 x 是向 x 轴方向平移位移,参数 y 是向 y 轴方向平移位移。

2. 缩放 scale(x,y)

对图形进行指定比例的放大或缩小,图形的位置不变。参数 x 是 x 坐标轴缩放比例,参数 y 是 y 坐标轴缩放比例。

3. 旋转 rotate(angle)

以画布的原点坐标(0,0)为参照点进行图形旋转,图形的大小、形状不变。参数 angle 是坐标轴旋转的角度(角度变化模型和画圆的模型一样)。

4. 变形 transform()

使用数学矩阵多次叠加形成更复杂的变化。可以调用 transform()方法对绘制的 canvas 图形进行变形,语法如下:

```
context.transform(m11, m12, m21, m22, dx, dy);
```

假定点(x, y)经过变形后变成了(X, Y),则变形的转换公式如下:

$$X = m11 \times x + m21 \times y + dx$$

$$Y = m12 \times x + m22 \times y + dy$$

【例 5-9】 图形的变换例子。

WXML 代码如下:

```
<canvas canvas - id = "myCanvas" style = "width: 300px; height: 300px;" ></canvas >
```

JS 代码如下:

```
Page({
    onLoad: function(options) {
    //创建画布上下文
```

```
    var context = wx.createCanvasContext('myCanvas')
    context.save();                                          //保存了当前 context 的状态
    context.fillStyle = "#EEEEFF";
    context.fillRect(0, 0, 400, 300);
    context.fillStyle = "rgba(255,0,0,0.1)";
    context.fillRect(0, 0, 100, 100);                        //正方形
    //平移 1 缩放 2 旋转 3
    context.translate(100, 100);                             //坐标原点平移(100, 100)
    context.scale(0.5, 0.5);                                 //x、y 轴是原来的一半
    context.rotate(Math.PI / 4);                             //旋转 45 度
    context.fillRect(0, 0, 100, 100);                        //平移、缩放、旋转后的正方形
    context.restore();                                       //恢复之前保存的绘图状态
    context.beginPath();                                     //开始绘图路径
    context.arc(200, 50, 50, 0, 2 * Math.PI, false);         //绘制圆
    context.stroke();
    context.fill();
    context.draw();
  },
})
```

运行结果如图 5-10 所示。

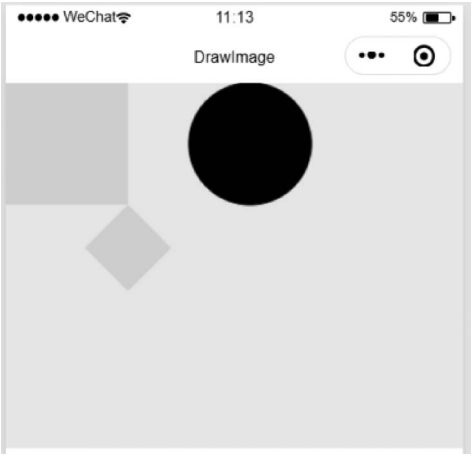


图 5-10　图形的变换

5.3　canvas 动画实例

在开发在线游戏时，绘制动画是非常重要的。本节介绍一个使用 canvas 实现的动画实例——游戏人物的跑步动画。

5.3.1　动画的概念及原理

1. 动画

动画是通过一幅幅静止的，内容不同的画面（即帧）快速播放使人们在视觉上产生运动的感觉，这是利用了人类眼睛的视觉暂留原理。利用人的这种生理特性可制作出具有高度想象力和表现力的动画影片。

2. 原理

人们在看画面时，画面会在大脑视觉神经中停留大约 1/24 秒，如果每秒更替 24 个画面或更多，那么前一个画面还没在人脑中消失之前，下一个画面进入人脑，人们就会觉得画面动起来了，它的基本原理与电影、电视一样，都是视觉原理。

在计算机上要实现动画效果，除了绘图外，还需要解决下面两个问题。

(1) 定期绘图，也就是每隔一段时间就调用绘图函数进行绘图。动画是通过多次绘图实现的，一次绘图只能实现静态图像。

可以使用 setInterval()方法设置一个定时器，语法如下：

```
setInterval(函数名,时间间隔)
```

时间间隔的单位是毫秒(ms)，每经过指定的时间间隔，系统都会自动调用指定的函数完成绘画。

(2) 清除先前绘制的所有图形。物体已经移动开来，可原来的位置上还保留先前绘制的图形，这样当然不行。解决这个问题最简单的方法是使用 clearRect(x, y, width, height)方法清除画布中的指定区域内容。

图 5-11 是一个方向(一般都是 4 个方向)的跑步动作序列图。假如想获取一个姿态的位图，可利用 canvas 的上下文对象的 drawImage(image, sourceX, sourceY, sourceWidth, sourceHeight, destX, destY, destWidth, destHeight)将原位图上某个区域(sourceX, sourceY, sourceWidth, sourceHeight)，复制到目标区域的(destX, destY)坐标点处，显示大小为(宽 destWidth，高 destHeight)。

图 5-11 一个方向的跑步动作序列

【例 5-10】 实现从跑步动作序列 Snap1.jpg 文件中截取第 3 个动作(帧)。

分析：在 Snap1.jpg 文件中，每个人物动作的大小 60 像素×80 像素，所以截取源位图的 sourceX=120，sourceY=0，sourceWidth=60，sourceHeight=80 就是第 3 个动作(帧)。

WXML 代码如下：

```
<canvas canvas-id="myCanvas" style="width: 300px; height: 300px;" ></canvas>
```

JS 代码如下：

```
Page({
onLoad: function() {
  //创建画布上下文
  var ctx = wx.createCanvasContext('myCanvas')
  //从原图(120, 0)位置开始截取中间一块宽 60×高 80 的区域,原大小显示在屏幕(0,0)处
  ctx.drawImage("people.jpg", 120, 0, 60, 80, 0, 0, 60, 80);
  ctx.draw();
  }
})
```

运行结果如图 5-12 所示，在页面上仅仅显示第 3 个动作。

图 5-12　静态显示第 3 个动作

5.3.2　游戏人物的跑步动画

【例 5-11】 实现游戏人物的跑步动画。

首先定义一个 canvas 元素，画布的长和宽都是 300，WXML 代码如下：

```
<canvas canvas - id = "myCanvas" style = "width: 300px; height: 300px;" ></canvas>
```

在 JavaScript 代码中定义一个 init()函数，并设置定时器，代码如下：

```
var x = 300;
var n = 0;                          // 计数器
Page({
  onLoad: function () {
    this.init();
  },
  init: function(){
    setInterval(this.draw, 100);// 定时器，每 0.1 秒执行一次 draw()函数
  },
```

使用了定时器，每隔 100 毫秒就会在 Snap1.jpg 图片截取一张 60 像素×80 像素大小的小图并绘制出来，且每次向左移 15 像素，直到最左端时重新从右侧开始，不停循环，就可见游戏人物在屏幕上不停地奔跑。

下面分析 draw()函数实现。例 5-10 中仅仅显示人物第 3 个动作，而为了实现动画需要 clearRect(x, y, width, height)不断清除先前绘制的动作图形，再绘制后续的动作。所以需要一个计数器 n，记录当前绘制第几动作(帧)了。

```
  draw: function() {
    var ctx = wx.createCanvasContext('myCanvas')      //创建画布上下文对象
    ctx.clearRect(0, 0, 300, 300);                    // 清除 canvas 画布
    //从原图(60 * n)位置开始截取中间一块宽 60 × 高 80 的区域，显示在屏幕(x,0)处
    ctx.drawImage('people.jpg', 60 * n, 0, 60, 80, x, 0, 60, 80);
    if (n >= 8) {
      n = 0;
    } else {
      n++;
    }
    if (x >= 0) {
      x = x - 30;                                     //前移 30 像素
    } else {
      x = 300;                                        //回到右侧
    }
```

```
    ctx.draw();
  },
})
```

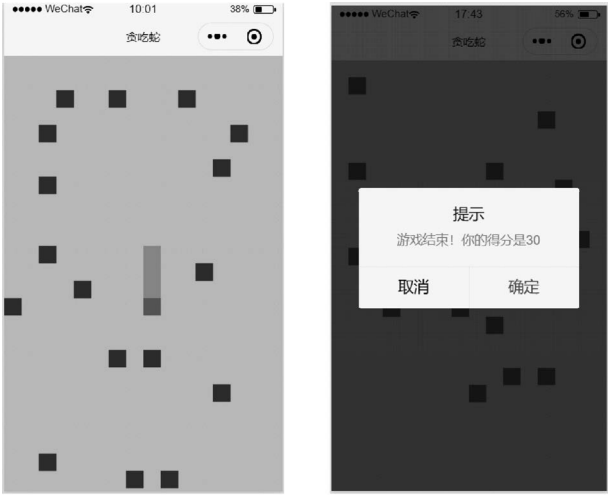
运行的结果是一个游戏人物不停重复地从右侧跑到左侧的动画。

扫一扫

视频讲解

5.4 拓展案例——贪吃蛇游戏

在该游戏中，玩家操纵一条贪吃的蛇在长方形场地里行走，贪吃蛇按玩家所滑动的方向键折行，蛇头吃到食物（豆）后，分数加 10 分，蛇身会变长，如果贪吃蛇碰上墙壁（出了边界）游戏就结束并显示得分。游戏运行界面如图 5-13 所示。



(a) 游戏运行　　(b) 游戏结束

图 5-13　贪吃蛇游戏运行界面

5.4.1 程序设计的思路

游戏画面看成由 W×H 个方格组成，每个方格大小为 20×20。豆和组成蛇的块均在屏幕上占据一个方格。设计时食物（豆）和组成蛇的块抽象成对象。

例如蛇头对象：

```
//蛇头对象
var snakeHead = {
  x: 0,                  //坐标位置
  y: 0,
  color: "#ff0000",      //蛇头颜色
  w: 20,                 //蛇头宽度
  h: 20                  //蛇头高度
}
```

组成蛇的块就是蛇头这样的对象，一条蛇可以看成由许多“块”（或称节）拼凑成的，块是蛇身上最小的单位。组成蛇的块使用 snakeBody 数组保存，其中包括蛇头这样的对象。使

用定时器移动蛇头以及 snakeBody 数组保存其他的“块”对象。

为了绘制蛇的移动效果，定时事件中首先根据用户滑动方向计算出蛇头新位置，将新位置蛇头加入 snakeBody 数组，同时将蛇尾 snakeBody[0]元素删去，重新绘制 snakeBody 数组里的对象即可达到蛇不断前移的效果。如果计算出蛇头新位置和食物(豆)位置相同(即碰到食物)，则蛇尾 snakeBody[0]元素不被删去，则达到蛇身增长的效果同时得分加 10 分。

同样食物(豆)对象如下：

```
foods.push({
  x: x,              //坐标位置
  y: y,
  w: 20,             //食物(豆)宽度
  h: 20,             //食物(豆)高度
  color: color       //食物(豆)颜色
})
```

通过判断蛇头对象 snakeHead 的坐标与食物(豆)对象坐标从而判断出是否吃到食物。

游戏是否结束，判断十分简单，判断蛇头是否达到边界即可。代码如下：

```
if (snakeHead.x > W * 20 || snakeHead.y > H * 20 ||
  snakeHead.x < 0 || snakeHead.y < 0) {
      console.log( "游戏结束!你的得分是" + score);
}
```

5.4.2　获取屏幕大小

本游戏中设置游戏场地的高度和宽度，需要获取屏幕大小。微信小程序提供的 wx.getSystemInfo()方法可以获取到设备的常用信息，如手机型号、设备像素比、屏幕宽高等，最常用的就是屏幕宽高。例如：

```
var m1 = wx.getSystemInfoSync().screenWidth;
var m2 = wx.getSystemInfoSync().screenHeight; //screenHeight 含有标题栏高度
```

微信小程序使用全新尺寸单位 rpx 实现屏幕自适应。其基本原理是无视设备原先的尺寸，统一规定屏幕宽度为 750rpx。rpx 可以根据屏幕宽度进行自适应。rpx 不是固定值，屏幕越大，1rpx 对应像素就越大。如在 iPhone 6 上，屏幕宽度为 375px，共有 750 个物理像素，则 750rpx = 375px = 750 物理像素，1rpx = 0.5px = 1 物理像素。

5.4.3　小程序中 this 和 that 的使用

this 是 JavaScript 语言的一个关键字，可以调用函数。当函数运行时，this 可以在函数内部使用，当函数使用场合发生变化时，this 的值也会发生变化。

在小程序开发中，小程序提供的 API 接口经常会有 success、fail 等回调函数来处理后续逻辑，当需要获取当前页面对象来对视图层进行渲染时，this 只会指向调用函数的对象，如果想要获取页面的初始数据，在回调函数里面就不能使用 this.data 来获取，同时也不能使用 this.setData()函数来更新数据，通过 var that = this；将 this 指向的对象复制到 that 当中才可以执行后续操作。

例如在移动蛇的函数 move()中,使用 setInterval()定时函数要注意 this 的指向。

```
move: function() {              //移动蛇的函数
  var that = this;              //注意此处
  this.data.timer = setInterval(function() {
    //获取画布的上下文
    var context = wx.createCanvasContext('snakeCanvas', this);
    that.drawWall(context); //绘制墙壁
    …                           //省略
    }, 600)                     //600 毫秒定时事件发生一次从而移动一次
}
```

5.4.4 JavaScript 数组操作

在游戏中定义蛇头对象 snakeHead,蛇身对象数组 snakeBody。游戏中需要对蛇身不断增加或者删除,这里主要使用数组的元素增删实现。

```
var snakeHead = {
  x: 100,
  y: 100,
  color: "#ff0000",            //蛇头红色
  w: 20,
  h: 20
}
var snakeBody = [];            //蛇身
```

例如蛇身增加一个节点,就是蛇身 snakeBody 数组添加一个对象元素。代码如下:

```
snakeBody.push({
  x: snakeHead.x,
  y: snakeHead.y,
  w: 20,
  h: 20,
  color: "#00ff00"
})
```

蛇尾是 snakeBody[0]元素,所以删除蛇尾就是将 snakeBody[0]元素从数组删去。代码如下:

```
snakeBody.shift();          //删去首元素即 snakeBody[0]元素
```

5.4.5 程序设计步骤

1. index.wxml 视图文件

index.wxml 文件内部仅添加 canvas 画布,并设置触屏事件函数。

```
<canvas canvas-id="snakeCanvas"
  style="width:100%;height:100%;background-color:#ccc"
  bindtouchstart="touchStart" bindtouchmove="touchMove" bindtouchend="touchEnd">
</canvas>
```

样式文件设置页面 page 占据整个屏幕。

```
/ ** index.wxss ** /
page {
  width: 100 % ;
  height: 100 % ;
}
```

2. 设计脚本(index.js)

定义触屏,计算位移所需按下坐标(startx, starty)、手指滑动时坐标(movex, movey)和计算手指滑动时 X 方向和 Y 方向的位移量(x, y)。

```
//手指按下的坐标
var startx = 0;
var starty = 0;
//手指在 canvas 移动时的坐标
var movex = 0;
var movey = 0;
//差值
var x = 0;          //X 方向位移
var y = 0;          //Y 方向位移
```

定义蛇相关变量和对象。

```
var direction = 'right';          //移动方向,'right'是向右移动
var score = 0; //游戏得分
var snakeDirection = 'right';     //方向
//蛇头对象
var snakeHead = {
  x: 100,
  y: 100,
  color: "#ff0000",               //蛇头红色
  w: 20,
  h: 20
}
var snakeBody = [];               //蛇身
var one = null;
var foods = [];                   //食物数组
var W = 15;                       //游戏场地大小宽度和高度
var H = 25;
```

collide2(obj1, obj2)函数实现蛇头对象与食物的碰撞检测,这里比较简单,仅比较坐标相同即可。

```
//蛇碰到食物
function collide2(obj1, obj2) {
  var l1 = obj1.x;
  var t1 = obj1.y;
  var l2 = obj2.x;
  var t2 = obj2.y;
  if (l1 == l2 && t1 == t2)
    return true;
  else
    return false;
}
```

页面中触屏事件计算出移动方向。

```
Page({
  data: {
    timer: '',              //定时器
  },
  touchStart: function(e) {//触屏开始
    startx = e.touches[0].x;
    starty = e.touches[0].y;
  },
  touchMove: function(e) {  //触屏滑动
    movex = e.touches[0].x;
    movey = e.touches[0].y;
    x = movex - startx;
    y = movey - starty;
  },
  touchEnd: function() {    //触屏结束
    console.log(x);
    console.log(y);
    if (Math.abs(x) > Math.abs(y) && x > 0) {
      direction = 'right';
      console.log('right');
    } else if (Math.abs(x) > Math.abs(y) && x < 0) {
      direction = 'left';
      console.log('left');
    } else if (Math.abs(x) < Math.abs(y) && y < 0) {
      direction = 'up';
      console.log('up');
    } else {
      direction = 'down';
      console.log('down');
    }
    snakeDirection = direction;
  },
```

drawWall：function(context)函数绘制墙壁。

```
drawWall: function(context) {//绘制墙壁
  for (var x = 0; x <= W; x++)
  {
    context.drawImage('wall.gif', x * 20, 0, 20, 20);
    context.drawImage('wall.gif', x * 20, H * 20, 20, 20);
  }
  for (var y = 0; y <= H; y++) {
    context.drawImage('wall.gif', 0, y * 20, 20, 20);
    context.drawImage('wall.gif', W * 20,y * 20 , 20, 20);
  }
},
```

游戏开始时，获取屏幕大小计算出游戏场地宽度 W 和高度 H，产生 20 个食物并开始移动蛇。

```
/**
 * 生命周期函数——监听页面加载
 */
onLoad: function(options) {
  var m1 = wx.getSystemInfoSync().screenWidth;
  W = Math.floor(m1 / 20) - 1;
  var m2 = wx.getSystemInfoSync().screenHeight;
  H = Math.floor(m2 / 20) - 4;          //screenHeight 含有标题栏高度
  console.log(m1 + ":" + m2);
  console.log(W + ":" + H);
  for (var i = 0; i < 20; i++) {
     this.creatFood();
  }
  this.move();
},
```

creatFood ()函数向 foods 数组里添加食物对象。

```
creatFood: function() {      //生成食物函数
  var x = Math.floor(Math.random() * (W - 1) + 1) * 20;
  var y = Math.floor(Math.random() * (H - 1) + 1) * 20;
  var w = 20;
  var h = 20;
  var color = "#0000ff";
  foods.push({
    x: x, y: y,
    w: 20, h: 20,
    color: color
  })
},
```

move()函数为了绘制蛇的移动效果，定时事件中首先根据用户滑动方向计算出蛇头新位置，将新位置蛇头加入 snakeBody 数组，同时将蛇尾 snakeBody[0]元素删去，重新绘制 snakeBody 数组里的对象即可达到蛇不断前移的效果。如果计算出蛇头新位置和食物(豆)位置相同(即碰到食物)，则蛇尾 snakeBody[0]元素不被删去，则达到蛇身增长的效果同时得分加 10 分。

```
move: function() {              //移动蛇的函数
  var that = this;
  this.data.timer = setInterval(function() {
    //获取画布的上下文
    var context = wx.createCanvasContext('snakeCanvas', this);
    that.drawWall(context);    //绘制墙壁
    //蛇头新位置
    switch (snakeDirection) {
      case "left":
        snakeHead.x -= 20;
        break;
      case "right":
        snakeHead.x += 20;
        break;
```

```
      case "up":
        snakeHead.y -= 20;
        break;
      case "down":
        snakeHead.y += 20;
        break;
    }
    //是否碰壁
    if (snakeHead.x >= W * 20 || snakeHead.y >= H * 20 ||
      snakeHead.x <= 0 || snakeHead.y <= 0) {
      clearInterval(that.data.timer);              //清除定时器
      wx.showModal({
        title: '提示',
        content: "游戏结束!你的得分是" + score,
        success: function(res) {
          if (res.confirm) {
            console.log('用户单击确定')
            that.newGame();                        //重新开始
          } else if (res.cancel) {
            console.log('用户单击取消');
          }
        }
      });
      console.log("游戏结束!你的得分是" + score);
    }
    //将新位置蛇头加入 snakeBody 数组
    snakeBody.push({
      x: snakeHead.x,
      y: snakeHead.y,
      w: 20,
      h: 20,
      color: "#00ff00"
    })
    var collideBoolean = false;                    //没碰到为 false
    //是否碰到食物
    for (var i = 0; i < foods.length; i++) {
      one = foods[i]
      if (collide2(snakeHead, foods[i])) {
        collideBoolean = true;
        foods.splice(i, 1);                        //删除碰到食物
        console.log("碰到食物 foods[i].x");
        console.log(one.x);
        score += 10;
        that.creatFood();
      }
    }
    if (snakeBody.length > 2 && collideBoolean == false) {
      snakeBody.shift();                           //删去首元素蛇尾
    }
    //以下开始绘制蛇身(含蛇头)和所有食物
```

```
      //绘制蛇身(含蛇头)
      for (var i = 0; i < snakeBody.length; i++) {
        var one = snakeBody[i];            //一节蛇身
        context.setFillStyle(one.color);
        context.beginPath();
        context.rect(one.x, one.y, one.w, one.h);
        context.closePath();
        context.fill();
      }
      context.setFillStyle(snakeHead.color);
      context.beginPath();
      context.rect(snakeHead.x, snakeHead.y, snakeHead.w, snakeHead.h);
      context.closePath();
      context.fill();
      //绘制食物
      for (var i = 0; i < foods.length; i++) {
        var one = foods[i]; //一个食物
        context.setFillStyle("#0000ff");
        context.beginPath();
        context.rect(one.x, one.y, one.w, one.h);
        context.closePath();
        context.fill();
      }
      context.draw();
    }, 600) //600 毫秒定时事件发生一次从而移动一次
  }
```

newGame()函数重新开始游戏。它清空蛇身和食物数组，重新产生 20 个食物，并设置蛇头位置并开始移动蛇。

```
    newGame: function(options) {
      snakeBody = [];          //清空蛇身
      foods = [];              //清空食物
      for (var i = 0; i < 20; i++) {
        this.creatFood();
      }
      snakeHead.x = 100;
      snakeHead.y = 100;
      this.move();
    },
  })
```

至此，贪吃蛇游戏编写完成。

习题 5

1. 设计井字棋游戏并能判断输赢。井字棋游戏在九宫格内进行，如果一方首先在某方向(横、竖、斜)连成三子，则获得胜利。游戏运行界面如图 5-14 所示。

2. 设计一个走迷宫游戏，并能判断是否走出迷宫。

图 5-14　井字棋游戏小程序