目录

[canvas 1](#_Toc6577)

[复习： 3](#_Toc15096)

[一、 canvas 6](#_Toc2765)

[二、 坐标系 8](#_Toc22338)

[2.1 API 9](#_Toc31029)

[三、 绘制图片 10](#_Toc3033)

[下午复习： 12](#_Toc32550)

[四、 状态的保存与恢复 15](#_Toc9607)

[五、 获取canvas上的像素信息 16](#_Toc27576)

[5.1 getImageData 16](#_Toc32406)

[5.2 putImageData 17](#_Toc8542)

[六、 融合 17](#_Toc17358)

[6.1 ctx.globalCompositeOperation 17](#_Toc9556)

# 复习：

本地存储：它是一个存储对象

对象名称： localStorage

它的生命周期是永久

API:

存储数据：

localStorage.setItem(key, value)

key: 要存储的数据名称

value: 要存储的数据

获取数据：

localStorage.getItem(key)

key: 要获取的数据

删除某一项数据：

localStorage.removeItem(key)

key: 要删除的数据

删除所有数据：

localStorage.clear()

除了本地存储之外还有一个存储对象

对象的名称： sessionStorage

它与本地存储的区别：生命周期

sessionStorage生命周期是从页面打开到关闭

多线程：

js是单线程，当js在执行的时候，页面是停止渲染的装态，只有当js执行完毕之后，页面才能渲染

在HTML5中提供了一个Worker函数用于开辟一个新的线程

使用方式:

1 要将计算的代码提取到一个js文件中

2 是初始化Worker函数，接收一个参数就是提取出来的js文件

消息传递：

开辟的额外的线程可以通过postMessage将结果推送到主线程中

主线程可以通过onmessage接收额外的线程推送的信息

WebSql:

前端数据库，本来是HTML5的规范，但是后来又不是

openDatabase(dbname, version, descript, size)

dbname: 数据库名称

version: 版本号

descript: 描述

size: 大小

该方法的作用：打开或者是创建数据库

当创建完成之后会返回一个数据库对象

通过对象调用transaction并且传递一个回调函数，函数中的参数就是事务对象

通过ta调用事务对象原型中的executeSql执行sql语句

executeSql(sql, [], sucessCallback, failedCallback)

history:

histroy.forward:

用于加载历史记录列表中的下一个URL

调用该方法等价于点击了前进按钮或者是调用了history.go(1)

history.back:

该方法会加载历史记录列表中的前一个URL

调用该方法等价于点击了后退按钮或者是调用了history.go(-1)

pushState:

用于添加新的历史记录

使用方式：

history.pushState(obj, title, url)

replaceState:

用于替换当前的历史记录

使用方式：

history.replaceState(obj, title, url)

拖拽事件：

ondrag: 拖拽

ondragstart: 拖拽开始

ondragend: 拖拽结束

ondragenter: 拖拽进入

ondragleave: 拖拽离开

ondragover: 悬浮

ondrop: 丢弃事件，有一个bug存在，ondragover的默认行为导致ondrop事件不能执行，如果想要给一个元素添加ondrop事件，必须给该元素添加ondragover事件并且阻止默认行为

# canvas

canvas是HTML5中新增的一个标签，用于提供“画布”

获取画笔要通过js脚本来获取

canvas中的标准属性有width和height (例如id、class 都是通用标准属性)

width: canvas的宽

height: canvas的高

举例：

|  |
| --- |
| 1. <style type="text/css"> 2. \* { 3. margin: 0; 4. padding: 0; 5. } 6. canvas { 7. display: block; 8. margin: 0 auto; 9. background-color: red; 10. } 11. </style> 12. </head> 13. <body> |

样式：

|  |
| --- |
|  |

获取画笔的方式：

1 要获取canvas元素：

|  |
| --- |
| 1. // 获取元素 2. var canvas = document.getElementById("myCanvas"); |

2 通过canvas元素获取画笔

|  |
| --- |
| 1. // 通过canvas元素获取对应的画笔 2. var ctx = canvas.getContext("2d"); |

查看画笔：

|  |
| --- |
| 2. 方法: |

画圆案例：

|  |
| --- |
| 1. // 绘制圆 2. // 开启路径 3. ctx.beginPath(); 4. // 绘制弧 5. // ctx.arc(100, 100, 50, Math.PI / 2, Math.PI \* 2, true); 6. ctx.arc(100, 100, 50, 0, Math.PI \* 2); 7. // 闭合路径 8. ctx.closePath(); 9. // 填充 10. ctx.fill(); |

结果：

|  |
| --- |
|  |

# 坐标系

canvas是一块区域，用来显示图像的，自带了一个坐标系，默认是与元素定定位坐标系是一致的

x表示正方向向右，y表示正方向向下

举例：

|  |
| --- |
| 1. // 获取元素 2. var canvas = document.getElementById("myCanvas"); 3. // 获取画笔 4. var ctx = canvas.getContext("2d"); 5. // 改变文字样式 6. ctx.font = "20px 宋体"; 7. // 绘制x轴 8. line(5, 5, 780, 5); 9. line(775, 0, 780, 5); 10. line(775, 10, 780, 5); 11. // 绘制y轴 12. line(5, 5, 5, 580); 13. line(0, 575, 5, 580); 14. line(10, 575, 5, 580); 15. // 绘制文字 16. ctx.fillText("X轴", 740, 40); 17. ctx.fillText("Y轴", 30, 550); 18. // 封装一个函数，传递两个点，绘制两个点之间的线段 19. function line(x1, y1, x2, y2) { 20. // 开启路径 21. ctx.beginPath(); 22. // 移动画笔到某个位置 23. ctx.moveTo(x1, y1); 24. // 产生路径 25. ctx.lineTo(x2, y2); 26. // 闭合路径 27. ctx.closePath(); 28. // 描边 29. ctx.stroke(); 30. } |

结果：

|  |
| --- |
|  |

## API

canvas用来显示图形的，大部分都是在操作路径，所以首先要开启路径

ctx.beginPath()

ctx.closePath(): 闭合路径，在闭合时候的点，和最开始时候的点之间会形成一条线段

ctx.fillRect(x, y, w, h)

作用：填充矩形

x: 当前坐标系的x点

y: 当前坐标系的y点

w: 矩形的宽

h: 矩形的高

ctx.strokeRect(x, y, w, h)

作用： 描边矩形

x: 当前坐标系的x点

y: 当前坐标系的y点

w：矩形的宽

h：矩形的高

ctx.clearRect(x, y, w, h)

作用：清除canvas上的一块区域

x: 区域的x点

y：区域的y点

w：区域的宽

h: 区域的高

ctx.arc(x, y, r, star, end, bool)

作用：绘制弧

x: 圆弧所在的圆心的x点

y：圆弧所在的圆心的y点

star: 圆弧的起始位置

end：圆弧的结束位置  
 bool: 决定绘制的方向， false: 顺时针， true: 逆时针

ctx.moveTo(x, y)

作用：移动画笔到某个位置

特点：不会产生路径

ctx.lineTo(x, y)

作用：移动画笔到某个位置

特点：会产生路径

ctx.fillText: 填充文字

ctx.font: 改变字体样式

ctx.lineWidth: 改变线宽

ctx.fillStyle: 改变填充色

ctx.strokeStyle: 改变描边色

# 绘制图片

在canvas中绘制图片的方式有三种： 1 插入图片 2 设置背景图 3 通过drawImage

drawImage方法使用方式有三种：

第一种是以原尺寸绘制图片：

使用方式:

ctx.drawImage(img, x, y)

img: 要绘制的图片

x: 以原尺寸将图片放在canvas中的x点

y：以原尺寸将图片放canvas上的y点

第二种使用方式：

缩放图片：

使用方式：

ctx.drawImage(img, x, y, w, h)

img: 要绘制的图片

x: 缩放后的图片的x点

y: 缩放后的图片的y点

w: 缩放后的图片的宽

y：缩放后的图片的高

第三种使用方式：

可以截取图片中的某个部分：

使用方式：

ctx.drawImage(img, img\_x, img\_y, img\_w, img\_h, canvas\_x, canvas\_y, canvas\_w, canvas\_h)

img: 要绘制的图片

img\_x: 要截取的图片的x点

img\_y: 要截取的图片的y点

img\_w: 要截取的图片的宽

img\_h: 要截取的图片的高

canvas\_x: 将截取后的图片放在canvas中的x点

canvas\_y: 将截取后的图片放在canvas中的y点

canvas\_w 将截取后的图片放在canvas上的宽

canvas\_h: 将截取后的图片放在canvas中的高

## 下午复习：

canvas: 是HTML5新增的标签用于提供“画布”

画笔要通过js来获取

在canvas标签中的标准属性有width和height

width: canvas的宽

height: canvas的高

获取画笔的方式：

1 要获取canvas元素

2 要通过canvas元素来获取对应的画笔

坐标系：

canvas是用来显示图形的，所以自带了一个坐标系，与元素的定位坐标系是一致的，默认是在左上角

API:

ctx.beginPath(): 开启路径

ctx.closePath(): 闭合路径

ctx.fillRect(x, y, w, h)

作用：填充一个矩形

x: 矩形的x点

y: 矩形的y点

w: 矩形宽

h: 矩形高

ctx.strokeRect(x, y, w, h)

作用： 描边矩形

x: 矩形的x点

y: 矩形的y点

w: 矩形宽

h: 矩形高

ctx,clearRect(x, y, w, h)

作用：清除canvas中的一块区域

x: 区域的x点

y: 区域的y点

w: 区域的宽

h: 区域的高

ctx.moveTo(x, y)

作用：移动画笔到某个位置

特点：不会产生路径

ctx.lineTo(x, y)

作用：移动画笔到某个位置

特点：产生路径

ctx.fillStyle: 改变填充色

ctx.strokeSteyle: 改变描边色

ctx.fillText: 填充文字

ctx.font: 改变文字样式

ctx.lineWidth: 改变线宽

绘制图片：

可以通过drawImage方法绘制图片：

使用方式有三种：

第一种以原尺寸绘制图片

ctx.drawImage(img, x, y)

img: 要绘制的图片

x：以原尺寸将图片放在canvas中的x点

y: 以原尺寸将图片放在canvas中的y点

第二种使用方式：

可以缩放图片

ctx.drawImage(img, x, y, w, h)

img: 要绘制的图片

x：将缩放后的图片放在canvas中的x点

y: 将缩放后的图片放在canvas中的y点

w: 缩放后的图片的宽

h: 缩放后的图片的高

第三种使用方式：

可以截取图片中的某个部分：

ctx.drawImage(img, img\_x, img\_y, img\_w, img\_h, canvas\_x, canvas\_y, canvas\_w, canvas\_h)

img: 绘制的图片

img\_x: 要截取的图片的x点

img\_y: 要截取后的图片的y点

img\_w: 将要截取的图片的宽

img\_h: 将要截取的图片的高

canvas\_x: 将截取后的图片放在canvas中的x点

canvas\_y: 将截取后的图片放在canvas中的y点

canvas\_w: 将截取后的图片放在canvas上的宽

canvas\_h: 将截取后的图片放在canvas上的高

# 状态的保存与恢复

在canvas中绘制图形的时候，很有可能用到之前的状态，所以在canvas中提供了相应的API用于保存状态

举例：

|  |
| --- |
| 1. // 获取canvas 2. var canvas = document.getElementById("myCanvas"); 3. // 获取画笔 4. var ctx = canvas.getContext("2d"); 5. // 改变填充色 6. ctx.fillStyle = "blue"; 7. // 绘制矩形 8. ctx.fillRect(0, 0, 100, 100); 9. // 保存状态 10. ctx.save(); 11. // 改变填充色 12. ctx.fillStyle = "orange"; 13. // 绘制矩形 14. ctx.fillRect(100, 100, 100, 100); 15. // 恢复状态 16. ctx.restore(); 17. // 绘制矩形 18. ctx.fillRect(200, 200, 100, 100); |

结果：

|  |
| --- |
|  |

save可以使用多次，每save一次就相当于把canvas的状态复制一份，每restore一次就用掉一份

每save一次就好比是给子弹夹装了一颗子弹

每restore就好比是扣了一次扳机

# 获取canvas上的像素信息

## 5.1 getImageData

该方法可以获取到canvas中的像素信息

使用方式：

ctx.getImageData(x, y, w, h)

x: 获取的矩形的x点

y: 获取的矩形的y点

w: 获取的矩形的宽

h: 获取的矩形的高

特点：需要服务器环境支持

举例：

|  |
| --- |
| 1. // 获取canvas 2. var canvas = document.getElementById("MyCanvas"); 3. // 获取画笔 4. var ctx = canvas.getContext("2d"); 5. // 创建一个图片元素 6. var img = new Image(); 7. // 设置路径 8. img.src = "imgs/head\_pic.jpg"; 9. // 添加onload事件 10. img.onload = function() { 11. ctx.drawImage(this, 0, 0, 1024, 650); 12. // 获取像素 13. var imgData = ctx.getImageData(0, 0, 1024, 650); 14. console.log(imgData); 15. } |

结果：

|  |
| --- |
|  |

## 5.2 putImageData

该方法用于将修正之后的像素对象放回到canvas中

使用方式：

ctx.putImageData(imgData, x, y)

imgData：修正之后的像素信息对象

x: 放在canvas上的x点

y: 放在canvas上的y点

举例：

|  |
| --- |
| 1. // 去掉红色 2. red.onclick = function() { 3. // 获取像素 4. var imgData = ctx.getImageData(0, 0, 1024, 650); 5. // console.log(imgData); 6. for (var i = 0; i < imgData.data.length; i += 4) { 7. imgData.data[i] = 0; 8. } 9. // 要将修正之后的像素信息对象重新放回到canvas中 10. ctx.putImageData(imgData, 0, 0); 11. } |

# 融合

所谓融合，就是新图形和原有图形之间的覆盖关系

默认情况下，新图性覆盖原有图形

## 6.1 ctx.globalCompositeOperation

source-over 默认。在目标图像上显示源图像。

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。

source-in 在目标图像中显示源图像。只有目标图像内的源图像部分会显示，目标图像是透明的。

source-out 在目标图像之外显示源图像。只会显示目标图像之外源图像部分，目标图像是透明的。

destination-over 在源图像上方显示目标图像。

destination-atop 在源图像顶部显示目标图像。源图像之外的目标图像部分不会被显示。

destination-in 在源图像中显示目标图像。只有源图像内的目标图像部分会被显示，源图像是透明的。

destination-out 在源图像外显示目标图像。只有源图像外的目标图像部分会被显示，源图像是透明的。

lighter 显示源图像 + 目标图像。

copy 显示源图像。忽略目标图像。

xor 使用异或操作对源图像与目标图像进行组合。

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |