Day01:

HTML5 新特性 -- Unit01

1. 音频与视频的标签

1.1 视频格式

浏览器支持的视频格式：mp4、webm、ogg

浏览器	MP4	WEBM	OGG
IE9+	YES	NO	No
Chrome	YES	YES	YES
Firefox	NO	YES	YES
Safari	YES	NO	NO
Opera	NO	YES	YES

1.2 视频标签

· 简捷语法

```
<video src="视频文件 URL 地址" width="宽度" height="高度">
    内容区域展示当浏览器不支持此视频格式时 显
```

示的提示文本

</video>

· 标准语法

```
<video width="宽度" height="高度">
    <source src="视频文件 URL 地址"/>
    ...
        内容区域展示当浏览器不支持此视频格式时显示的提示文本
</video>
```

1.3 视频属性

· controls，布尔属性，用于控制是否显示视频播放控件

· autoplay，布尔属性，用于控制是否自动播放视频(需与 muted 属性组合使用)

//为了防止页面载入时播放的文本吓到用户

· muted，布尔属性，用于控制视频是否静音播放

· loop，布尔属性，用于控制视频是否循环播放

· poster,海报帧的 URL //是指加载时视频的初始帧写的是帧路径

· preload,控制视频的预载入方式

· auto,浏览器尽可能的下载视频文件(播放流畅，但会造成网络流量的浪费)

· none，不缓存视频，尽可能的减少流量的浪费

· metadata,只加载视频的时长、宽度、高度等信息

1.4 音频格式

浏览器支持的音频格式：mp3、wav、ogg

浏览器	MP3	WAV	OGG
IE9+	YES	NO	No
Chrome	YES	YES	YES
Firefox	NO	YES	YES
Safari	YES	YES	NO
Opera	NO	YES	YES

1.5 音频标签

· 简捷语法

```
<audio src="音频文件 URL">
    浏览器不支持音频的提示文本
</audio>
```

· 标准语法

```
<audio>
    <source src="音频文件 URL"/>
    ...
    浏览器不支持音频的提示文本
</audio>
```

1.6 音频属性

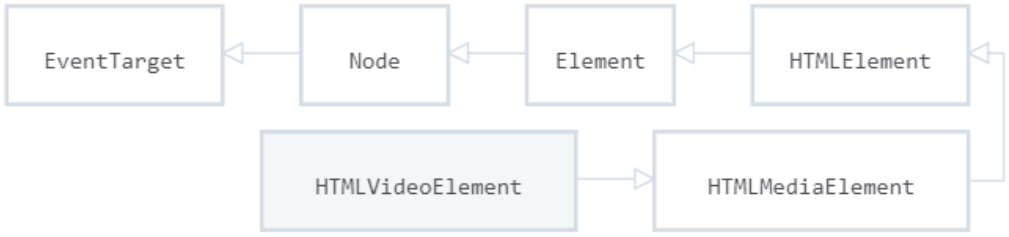
音频属性与音频属性是一样的 唯一不同的是

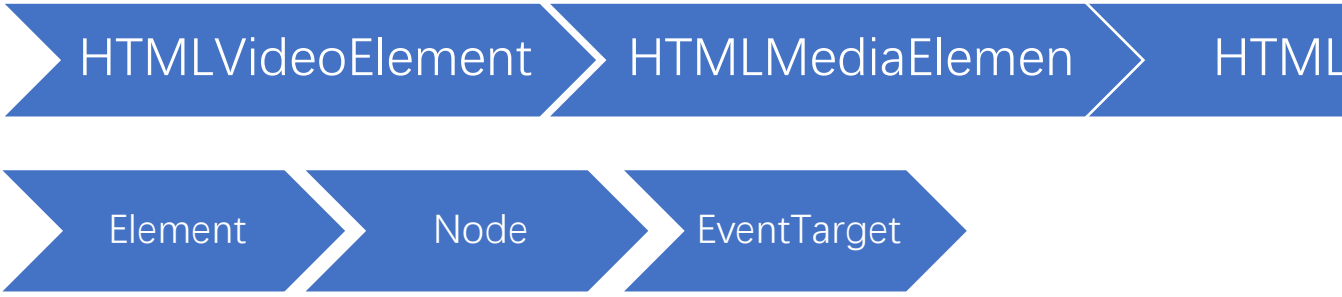
没有视频的 [poster,海报帧的 URL //是指加载时视频的初始帧 写的是帧路径]

2.JavaScript 与音频和视频

2.1 HTMLVideoElement

HTMLVideoElement 接口提供用于操作视频对象的属性和方法，该接口继承自 HTMLMediaElement 和 HTMLElement。





· width 属性

width 属性用于获取/设置视频对象的宽度，语法结构是：

```
//获取 给变量赋值的过程
variable = HTMLVideoElement.width

//设置 给原有变量更改值的过程
HTMLVideoElement.width = value
```

获取视频元素的 width 属性时必须保证 HTML <video>标签中存在 width 属性 也就是说如果标签中没有设置就获取不到宽高

· height 属性

height 属性用于获取/设置视频对象的高度，语法结构是：

```
//获取 给变量赋值的过程
variable = HTMLVideoElement.height

//设置 给原有变量更改值的过程
HTMLVideoElement.height = value
```

标签中没有设置就获取不到宽高

· videoWidth 属性

videoWidth 属性用于获取视频对象原始宽度，语法结构是：

获取 给变量赋值的过程

```
variable = HTMLVideoElement.videoWidth
```

· videoHeight 属性

videoHeight 属性用于获取视频对象原始高度，语法结构是：

获取 给变量赋值的过程

```
variable = HTMLVideoElement.videoHeight
```

如果要获取视频对象的原始宽度和高度必须在 loadeddata 事件完成后才能使用，其示例代码如下：

loadeddate 事件

```
<script>
    //获取 HTMLVideoElement 对象          根据 ID 查找元素
    let videoEle = document.getElementById('video'); //查找 DOM 元素
    //loadeddata 事件代表是已经加载完成视频的第一帧
    //如果已经加载了视频的第一帧的话，那么原始宽度和高度肯定已经能够进行访问了
                                                    事件名
    videoEle.addEventListener('loadeddata',()=>{  //监听变化
        //输出视频的原始宽度和高度
        console.log(videoEle.videoWidth);
        console.log(videoEle.videoHeight);

    });
</script>
```

方式	语法
1.在 DOM 中绑定	a. html 中: <元素 on 事件名="事件处理函数()"> b. js 中: function 事件处理函数(){ … } c. 问题: 因为事件绑定随元素分散在网页的各个角 落，极其不便于维护
2.在 js 中绑定	a. 元素对象.on 事件名=function(){ … } b. 好处: 所有事件绑定都集中在 js 中，非常便于维 护 c. 问题: 一个元素的一个事件属性上，只能保留一 个事件处理函数。无法同时保留多个事件处理函数。 重复给一个元素的一个属性上赋值多个事件处理函数，

	结果只有最后一个事件处理函数才能留下来!
3.事件监听	a. 元素对象.addEventListener("事件名", 事件处理 函数) 添加 事件 监听对象 b. 强调: 如果使用 addEventListener, 事件名之前 不用加"on"。 因为原本 DOM 标准中规定的事件名都 是没有 on 的! 比如: click, focus, change, ...

· poster 属性

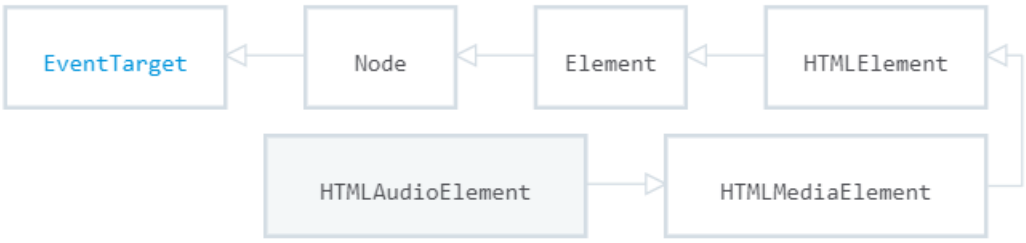
poster 属性用于获取/设置视频的海报帧, 其语法结构是:

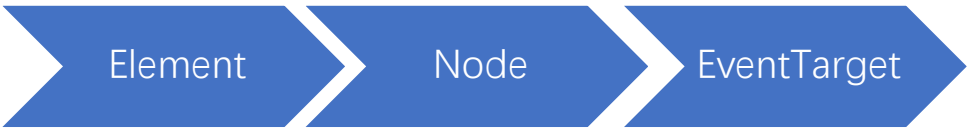
```
//获取 给变量赋值的过程
varible = HTMLVideoElement.poster
```

```
//设置 给原有变量更改值的过程
HTMLVideoElement.poster = value
```

2.2 HTMLAudioElement

HTMLAudioElement 接口提供用于操作音频对象的属性和方法，该接口继承自 HTMLMediaElement 和 HTMLElement。





· 构造函数

```
variable = new Audio([音频文件的 URL 地址])
//这里的 URL 地址为可选项 具体要看 HTML 标签中是否写了
```

2.3 HTMLMediaElement

HTMLMediaElement 接口提供用户操作媒体(音频和视频)的属性和方法。

2.3.1 属性

· autoplay

autoplay 属性用于获取/设置媒体对象是否自动播放，其语法结构是:

```
//获取 给变量赋值的过程
variable = HTMLMediaElement.autoplay

//设置 给原有变量更改值的过程
HTMLMediaElement.autoplay = boolean
```

· muted

muted 属性用于获取/设置媒体对象在播放时是否静音，其语法结构是:

```
//获取 给变量赋值的过程
```

```
variable = HTMLMediaElement.muted

//设置 给原有变量更改值的过程
HTMLMediaElement.muted = boolean
```

· controls

controls 属性用于获取/设置媒体对象在播放时是否显示播放控件，其语法结构是：

```
//获取 给变量赋值的过程
variable = HTMLMediaElement.controls

//设置 给原有变量更改值的过程
HTMLMediaElement.controls = boolean
```

· loop

loop 属性用于获取/设置媒体对象是否循环播放，其语法结构是：

```
//获取 给变量赋值的过程
```

```
variable = HTMLMediaElement.loop
```

```
//设置 给原有变量更改值的过程
HTMLMediaElement.loop = boolean
```

· src

src 属性用于获取/设置媒体文件的 URL 地址，其语法结构是：

```
//获取 给变量赋值的过程
variable = HTMLMediaElement.src
```

```
//设置 给原有变量更改值的过程
HTMLMediaElement.src = 媒体文件的 URL 地址
```

· volume

volume 属性用于获取/设置媒体的音量(取值范围来[0,1])，其语法结构是：

```
//获取 给变量赋值的过程
```

```
varaible = HTMLMediaElement.volume

//设置 给原有变量更改值的过程
HTMLMediaElement.volume = 值
```

· playbackRate

playbackRate 属性用于获取/设置媒体的播放速率，如果其值为 1.0，为正常速率，如果小于 1.0 则低于正常速率，否则高于正常速率。

```
//获取 给变量赋值的过程
varaible = HTMLMediaElement.playbackRate

//设置 给原有变量更改值的过程
HTMLMediaElement.playbackRate = 值
```

· paused

paused 属性用于获取媒体对象是否在暂停，语法结构是：

```
 获取 给变量赋值的过程
variable = HTMLMediaElement.paused
```

· ended

ended 属性用于获取媒体对象是否播放完毕，语法结构是：

```
variable = HTMLMediaElement.ended
```

· currentTime

currentTime 属性用于获取/设置媒体对象的当前播放时间(单位为秒)，其语法结构是：

```
//获取 给变量赋值的过程
varaible = HTMLMediaElement.currentTime

//设置 给原有变量更改值的过程
HTMLMediaElement.currentTime = 值
```

· duration

duration 属性用于获取媒体对象的总时长(单位为秒)，其语法结构是:

```
//获取 给变量赋值的过程
varaible = HTMLMediaElement.duration
```

2.3.2 方法

· play()

play()方法用于实现媒体的播放，其语法结构是:

```
HTMLMediaElement.play()
```

· pause()

pause()方法用于实现媒体的暂停，其语法结构是:

```
HTMLMediaElement.pause()
```

2.3.3 事件

· play

play 事件在媒体对象播放时触发，其语法结构是:

```
HTMLMediaElement.addEventListener('play',
()=>{
   ...
})

HTMLMediaElement.onplay = ()=>{
    ....
}
```

· pause

pause 事件在媒体对象暂停时触发，其语法结构是：

```
HTMLMediaElement.addEventListener('pause
',()=>{
   ...
})

HTMLMediaElement.onpause = ()=>{
    ....
}
```

· ended

ended 事件在媒体对象播放完毕后触发，其语法结构是：

```
HTMLMediaElement.addEventListener('ended',()=>{
   ...
})

HTMLMediaElement.onended = ()=>{
    ....
}
```

· loadeddata

loadeddata 事件在媒体文件的第一帧加载完成后被触发，其语法结构是：

```
HTMLMediaElement.addEventListener('loaded data',()=>{
   ...
})

HTMLMediaElement.onloadeddata = ()=>{
```

```
    ....
}
```

· timeupdate

timeupdate 事件在媒体对象的 currentTime 属性发生变化时调用，其语法结构是：

```
HTMLMediaElement.addEventListener('timeup
date',()=>{
   ...
})

HTMLMediaElement.ontimeupdate = ()=>{
    ....
}
```

js 总结：

2. 资料:

HTML 音频/视频

HTML 音频/视频 DOM 参考手册

HTML5 DOM 为 <audio> 和 <video> 元素提供了方法、属性和事件。

这些方法、属性和事件允许您使用 JavaScript 来操作 <audio> 和 <video> 元素。

HTML 音频/视频 方法

方法	描述
addTextTrack()	向音频/视频
canPlayType()	检测浏览器是
load()	重新加载音频
play()	开始播放音频
pause()	暂停当前播放

HTML 音频/视频属性

属性	描述

audioTracks	返回表示可用音频轨道的 A
autoplay	设置或返回是否在加载完成
buffered	返回表示音频/视频已缓冲部
controller	返回表示音频/视频当前媒体
controls	设置或返回音频/视频是否显
crossOrigin	设置或返回音频/视频的 CO
currentSrc	返回当前音频/视频的 URL。
currentTime	设置或返回音频/视频中的当
defaultMuted	设置或返回音频/视频默认是
defaultPlaybackRate	设置或返回音频/视频的默认
duration	返回当前音频/视频的长度
ended	返回音频/视频的播放是否已
error	返回表示音频/视频错误状态
loop	设置或返回音频/视频是否应
mediaGroup	设置或返回音频/视频所属的
muted	设置或返回音频/视频是否静

networkState	返回音频/视频的当前网络状
paused	设置或返回音频/视频是否暂
playbackRate	设置或返回音频/视频播放的
played	返回表示音频/视频已播放部
preload	设置或返回音频/视频是否应
readyState	返回音频/视频当前的就绪状
seekable	返回表示音频/视频可寻址部
seeking	返回用户是否正在音频/视频
src	设置或返回音频/视频元素的
startDate	返回表示当前时间偏移的 D
textTracks	返回表示可用文本轨道的 T
videoTracks	返回表示可用视频轨道的 V
volume	设置或返回音频/视频的音量

HTML 音频/视频事件

事件	描述

abort	当音频/视频的加载已放弃时触发。
canplay	当浏览器可以开始播放音频/视频时触发
canplaythrough	当浏览器可在不因缓冲而停顿的情况下
durationchange	当音频/视频的时长已更改时触发。
emptied	当目前的播放列表为空时触发。
ended	当目前的播放列表已结束时触发。
error	当在音频/视频加载期间发生错误时触发
loadeddata	当浏览器已加载音频/视频的当前帧时触
loadedmetadata	当浏览器已加载音频/视频的元数据时触
loadstart	当浏览器开始查找音频/视频时触发。
pause	当音频/视频已暂停时触发。
play	当音频/视频已开始或不再暂停时触发。
playing	当音频/视频在因缓冲而暂停或停止后已
progress	当浏览器正在下载音频/视频时触发。
ratechange	当音频/视频的播放速度已更改时触发。
seeked	当用户已移动/跳跃到音频/视频中的新

seeking	当用户开始移动/跳跃到音频/视频中的
stalled	当浏览器尝试获取媒体数据，但数据不
suspend	当浏览器刻意不获取媒体数据时触发。
timeupdate	当目前的播放位置已更改时触发。
volumechange	当音量已更改时触发。
waiting	当视频由于需要缓冲下一帧而停止时触

作业：

自定义视频播放器：

A.单击播放/暂停时实现播放/暂停(图标需要变化)

B.音击喇叭可实现静音与非静音的切换(图标需要变化)

C.在拖动音时滑块时控制音量的变化（提醒：如果拖动到最左侧时，要显示静音的图标）

D.将当前时长与总时长转换为分分:秒秒的形态,如00:33

Day02:

HTML5 新特性 -- Unit02

1.全屏模式

全屏模式可以让一个 Element 及其子元素占满整个屏幕。

· 进入全屏模式

```
//W3C 的建议
Element.requestFullscreen()
//Chrome、safari
Element.webkitRequestFullscreen()
//Firefox
Element.mozRequestFullScreen()
//Internet Explorer
Eelement.msRequestFullscreen()
```

· 退出全屏模式

```
//W3C 的建议
document.exitFullscreen()
//Chrome、safari
```

```
document.webkitExitFullscreen()
//Firefox
document.mozCancelFullScreen()
//Internet Explorer
document.msExitFullscreen()
```

· 获取全屏元素

```
//W3C 的建议
document.fullscreenElement
```

```
<body>
  <div id="container">
      <button onclick="enterFullscreen()">进入全屏模式
</button>
      <button onclick="exitFullscreen()">退出全屏模式
</button>
  </div>
 <script>
    function enterFullscreen(){
      //获取 DIV 元素
```

```
      let divEle = document.getElementById('container'
);
      //进入全屏模式
      divEle.requestFullscreen();
      console.log(document.fullscreenElement);
```

fullscreenElement 属性返回正处于全屏状态的 Element 节点，如果当前没有节点处于全屏状态，则返回 null。

```
    }
    function exitFullscreen(){
      //退出全屏模式
      document.exitFullscreen();
      console.log(document.fullscreenElement);
    }
  </script>
</body>
```

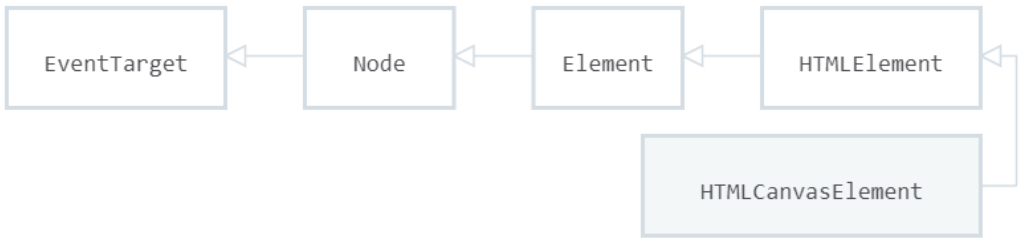
canvas 是通过 JavaScript 来绘制图形的 HTML 元素，其语法结构是：

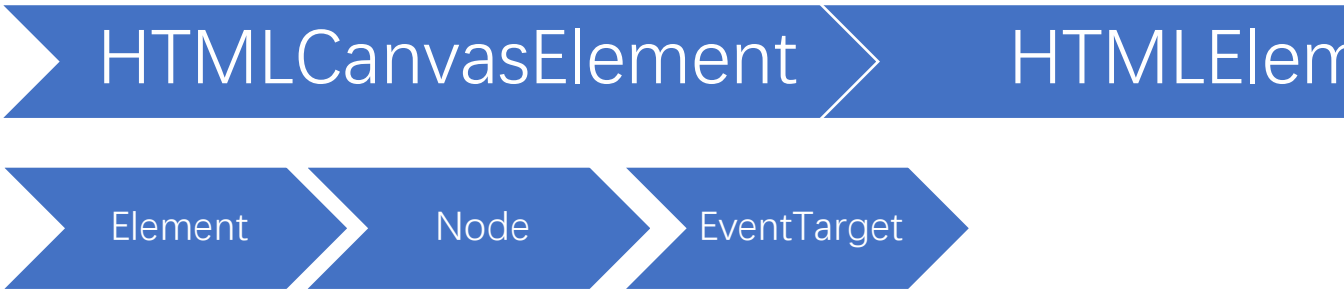
```
<canvas width="宽度" height="高度"></canvas>
```

默认情况下 <canvas> 元素没有边框和内容

2.1 HTMLCanvasElement

HTMLCanvasElement 接口提供处理<canvas>元素的属性和方法。该接口继承自 HTMLElement 接口。





· width 属性

width 属性用于获取/设置<canvas>元素的宽度，其语法结构是：

```
//获取 给变量赋值的过程
```

```
variable = HTMLCanvasElement.width

//设置 给原有变量更改值的过程
HTMLCanvasElement.width = value
```

· height 属性

height 属性用于获取/设置<canvas>元素的高度，其语法结构是：

```
//获取 给变量赋值的过程
variable = HTMLCanvasElement.height

//设置 给原有变量更改值的过程
HTMLCanvasElement.height = value
```

· getContext()方法

getContext()方法用于获取<canvas>元素的上下文，如果没有定义上下文则返回 null，其语法结构是：

HTMLCanvasElement.getContext(contextType)

contextType 参数值可以为:

2d，将返回 CanvasRenderingContext2D 对象，二维上下文渲染对象

webgl,将返回 WebGLRenderingContext 对象，三维上下文渲染对象

3.CanvasRenderingContext2D 对象

<canvas>元素的坐标原点(0,0)位画布的左上角

笔记中用 ctx 代表

CanvasRenderingContext2D 对象

· strokeRect()方法

strokeRect()方法用于绘制描边矩形，其语法结构是:

坐标系　宽　　高

ctx.strokeRect(x,y,width,height)

· fillRect()方法

fillRect()方法用于绘制填充矩形，其语法结构是：

坐标系　宽　　高

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

· strokeStyle 属性

strokeStyle 属性用于获取/设置描边颜色，其语法结构是：

```
//获取 给变量赋值的过程
variable = ctx.strokeStyle

//设置 给原有变量更改值的过程
ctx.strokeStyle = 颜色值
```

定义和用法

strokeStyle 属性设置或返回用于笔触的颜色、渐变或模式。

默认值:	#000000
JavaScript 语法:	*context*.strokeStyle=*colo*

属性值

值	描述
color	指示绘图笔触颜色的 CSS 颜色值。
gradient	用于填充绘图的渐变对象（线性或
pattern	用于创建 pattern 笔触的 pattern

渐变实例

绘制一个矩形。使用渐变笔触：

JavaScript：

```
var
c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");

var
gradient=ctx.createLinearGradient(0,0,17
0,0);
gradient.addColorStop("0","magenta");
gradient.addColorStop("0.5","blue");
gradient.addColorStop("1.0","red");

// Fill with gradient
ctx.strokeStyle=gradient;
ctx.lineWidth=5;
ctx.strokeRect(20,20,150,100);
```

· fillStyle 属性

fillStyle 属性用于获取/设置填充颜色，其语法结构是：

```
//获取 给变量赋值的过程
variable = ctx.fillStyle

//设置 给原有变量更改值的过程
ctx.fillStyle = color
```

· strokeText()方法

strokeText()方法用于绘制描边文本，其语法结构是：

参数值

参数	描述
text	规定在画布上输出的文本。
x	开始绘制文本的 x 坐标位置（相对
y	开始绘制文本的 y 坐标位置（相对
maxWidth	可选。允许的最大文本宽度，以像

```
ctx.strokeText(text,x,y)
```

· fillText()方法

fillText()方法用于绘制填充文本，其语法结构是：

参数值

参数	描述
text	规定在画布上输出的文本。
x	开始绘制文本的 x 坐标位置
y	开始绘制文本的 y 坐标位置
maxWidth	可选。允许的最大文本宽度，

```
ctx.fillText(text,x,y)
```

· font 属性

font 属性用于获取/设置文本样式，其语法结构是：

```
//获取 给变量赋值的过程
variable = ctx.font
//设置 给原有变量更改值的过程
ctx.font = "字号 字体"
```

属性	描述
font	设置或返回文本内容的当前字体属性。
textAlign	设置或返回文本内容的当前对齐方式。
textBaseline	设置或返回在绘制文本时使用的当前文本

2.Canvas

1. 资料:

HTML 画布

描述

HTML5 <获得用于在画布上绘图的对象获得用于在画布上绘图的对象> 标签用于绘制图像（通过脚本，通常是 JavaScript）。

不过，<canvas> 元素本身并没有绘制能力（它仅仅是图形的容器） - 您必须使用脚本来完成实际的绘图任务。

getContext() 方法可返回一个对象，该对象提供了用于在画布上绘图的方法和属性。

本手册提供完整的 getContext("2d") 对象的属性和方法，可用于在画布上绘制文本、线条、矩形、圆形等等。

浏览器支持

Internet Explorer 9、Firefox、Opera、Chrome 和 Safari 支持 <canvas> 标签的属性及方法。

注意: Internet Explorer 8 及更早的 IE 版本不支持 <canvas> 元素。

颜色、样式和阴影

属性	描
fillStyle	设
strokeStyle	设
shadowColor	设
shadowBlur	设
shadowOffsetX	设
shadowOffsetY	设

方法	描
createLinearGradient()	创
createPattern()	在
createRadialGradient()	创

addColorStop()	规定渐变对象

线条样式

属性	描述
lineCap	设置或返回线条的
lineJoin	设置或返回两条线
lineWidth	设置或返回当前的
miterLimit	设置或返回最大斜

矩形

方法	描述
rect()	创建矩形。
fillRect()	绘制"被填充"的矩
strokeRect()	绘制矩形（无填充
clearRect()	在给定的矩形内清

路径

方法	描述
fill()	填充当前
stroke()	绘制已定
beginPath()	起始一条
moveTo()	把路径移
closePath()	创建从当
lineTo()	添加一个
clip()	从原始画
quadraticCurveTo()	创建二次
bezierCurveTo()	创建三次
arc()	创建弧/曲
arcTo()	创建两切
isPointInPath()	如果指定

转换

方法	描述
scale()	缩放当前绘图至更
rotate()	旋转当前绘图。
translate()	重新映射画布上的
transform()	替换绘图的当前转
setTransform()	将当前转换重置为

文本

属性	描述
font	设置或返回文本内
textAlign	设置或返回文本内
textBaseline	设置或返回在绘制

方法	描述

fillText()	在画布上绘制"被填充的"文本
strokeText()	在画布上绘制文本（无填充）
measureText()	返回包含指定文本宽度的对象

图像绘制

方法	描述
drawImage()	向画布上绘制图像、画布或视频。

像素操作

属性	描述
width	返回 ImageData
height	返回 ImageData
data	返回一个对象，其

方法	描述
createImageData()	创建新的、
getImageData()	返回 Imag

putImageData()	把图像数据（从指定的

合成

属性
globalAlpha
globalCompositeOperation

其他

方法	描述
save()	保存当前环境的状态。
restore()	返回之前保存过的路径状态和属
createEvent()	创建新的 Event 对象
getContext()	获得用于在画布上绘图的对象
toDataURL()	导出在 canvas 元素上绘制的图

Day03:

HTML5 新特性 -- Unit03

1.canvas

· textAlign 属性

textAlign 属性用于设置文本的水平对齐方式，其语法结构是：

```
ctx.textAlign = 'left|center|right'
```

· clearRect()方法

clearRect()方法用于擦除指定的矩形区域，其语法结构是：

```
ctx.clearRect(x,y,width,height)
```

· 路径

路径(path)将预先设置的坐标点顺序连接形成的图形。

路径的绘制步骤:

A.通过 ctx.beginPath()方法开始新的路径

B.通过 ctx.moveTo()方法指定路径的起点坐标

C.绘制基本路径内容(如 lineTo()方法用于绘制线段等)

D.通过 stroke()或 fill()方法完成路径的绘制

路径

方法	描述

fill()	填充当前绘图（路径）
stroke()	绘制已定义的路径。

· beginPath()方法

beginPath()方法用于新始一个新的路径，其语法结构是：

```
ctx.beginPath()
```

· moveTo()方法

moveTo()方法用于指定路径的起点，其语法结构是：

```
ctx.moveTo(x,y)
```

· lineTo()方法

lineTo()方法用于绘制线段，其语法结构是：

```
ctx.lineTo(x,y)
```

· arc()方法

arc()方法用于绘制圆弧路径，其语法结构是：

```
ctx.arc(x,y,radius,start_angle,end_angle)
```

圆弧的起点和终点用弧度表示。

弧度的计算公式为：角度 * Math.PI / 180

· stroke()方法

stroke()方法用于根据当前的描边样式来绘制路径，其语法结构是:

```
ctx.stroke()
```

2.window 对象

· requestAnimationFrame()方法

requestAnimationFrame()方法用于在浏览器中定时循环执行一个操作。其优点是:

A.该方法充利用显示器的刷新频率，不会出现卡顿、丢帧的现象;

B.如果当前选项卡没有被激话的话，动画将自动停止，以节省计算机资源

requestAnimationFrame()方法的语法结构是:

```
variable = window.requestAnimationFrame(callback)
```

· cancelAnimationFrame()方法

cancelAnimationFrame()方法用于清理由 requestAnimationFrame()方法返回的 requestId。其语法结构是:

```
window.cancelAnimationFrame(requestId)
```

Day04:

HTML5 新特性 -- Unit04

1.Vuex

1.1 什么是 Vuex?

Vuex 是一个专门为 Vue 应用程序开发的状态管理模式，它采用集中式的管理方式来管理 Vue 组件中共享的状态（数据）。

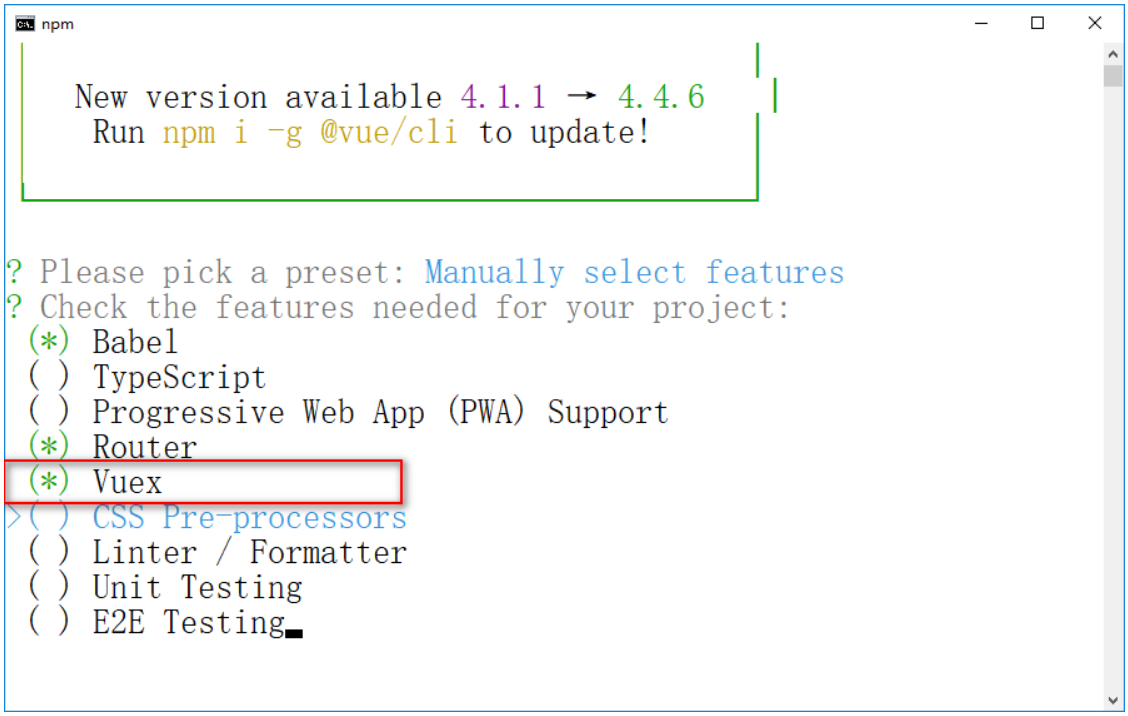
Vuex 的应用场景：

· 用户登录的状态

· 购物车的状态等

1.2 安装 Vuex

方式 1：在通过 vue create 命令创建脚手架项目时，选择 Vuex 插件



方式 2：通过 npm 进行安装

```
npm install --save vuex
```

Vuex 本质就是 Vue 的一个插件

Vuex 建议存储到 store 目录下的 index.js 中

1.3 基本用法

Vuex 的核心是 Store，其就是一个容器，包含了 Vue 应用中的状态。其基本结构是：

```
export default new Vuex.Store({
    state: {
    },
    mutations: {
    },
    actions: {
    },
    modules: {
    }
})
```

· state

state 定义了应用状态的数据结构，其数据类型可以为 string、number、boolean、array、object 等。示例代码如下：

```
state: {
    username:'Tom',
    age:23,
    sex:true,
    products:[
```

```
        {
            id:1,
            productName:'商品 1',
            salePrice:2389.99
        },
        {
            id:3,
            productName:'商品 2',
            salePrice:6999.99
        }
    ]
}
```

· Vue 组件访问 Vuex 中的状态

```
this.$store.state.xxx
```

示例代码如下：

```
<template>
    <div>
        <h1>访问 Vuex 的数据--页面 1</h1>
        <p>姓名:{{$store.state.username}}</
```

```
p>
        <p>年龄:{{$store.state.age}}</p>
        <p>性别:{{$store.state.sex ? '男' :
 '女'}}</p>
    </div>
</template>
<script>
    export default {
        mounted(){
            console.log(this.$store.state.a
ge);
        }
    }
</script>
```

getters

Vuex 允许在 store 中定义 getter(可以认为是 store 的计算属性)，getter 的返回值会自动缓存，只有其依赖的值发生了变化才会被重新计算，其语法结构是：

```
getters:{
    //获取商品的数量 -- 这些方法被认为是state的计算属性
    //state参数自动代表当前store的state(其实名称可以为任意合法名称)
    productNum(state){
     return state.products.length;
    }
},
```

```
<h2>商品列表中有{{this.$store.getters.productNum}}件,具体如下:</h2>
```

mutations

mutations 是用于改变状态的方法，也是唯一修改 state 的推荐方法，示例代码如下：

· 在页面组件中调用 mutations 中的方法

```
this.$store.commit('方法名称',[payload])
```

载荷

2.WebStorage

分类 sessionStorage 和 localStorage 两种存储机制。

length 属性

setItem()方法

getItem()方法

removeItem()方法

clear()方法

Day05:

HTML5 新特性 -- Unit05

1.Vuex

Vuex 的核心是 Store，其就是一个容器，包含了 Vue 应用中的状态。其基本结构是:

```
export default new Vuex.Store({
    state: {
    },
    getters:{
    },
    mutations: {
    },
    actions: {
    },
    modules: {
    }
})
```

· Actions

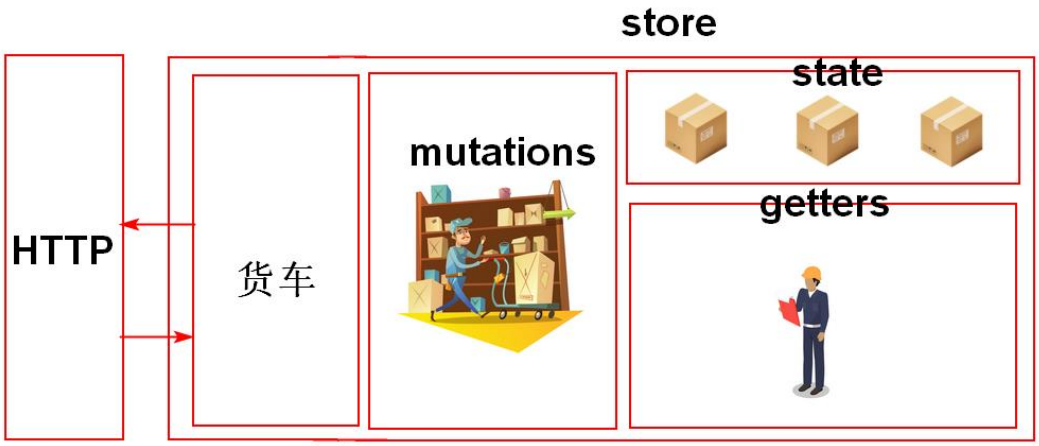
Actions 用于异步发送请求，示例代码如下：

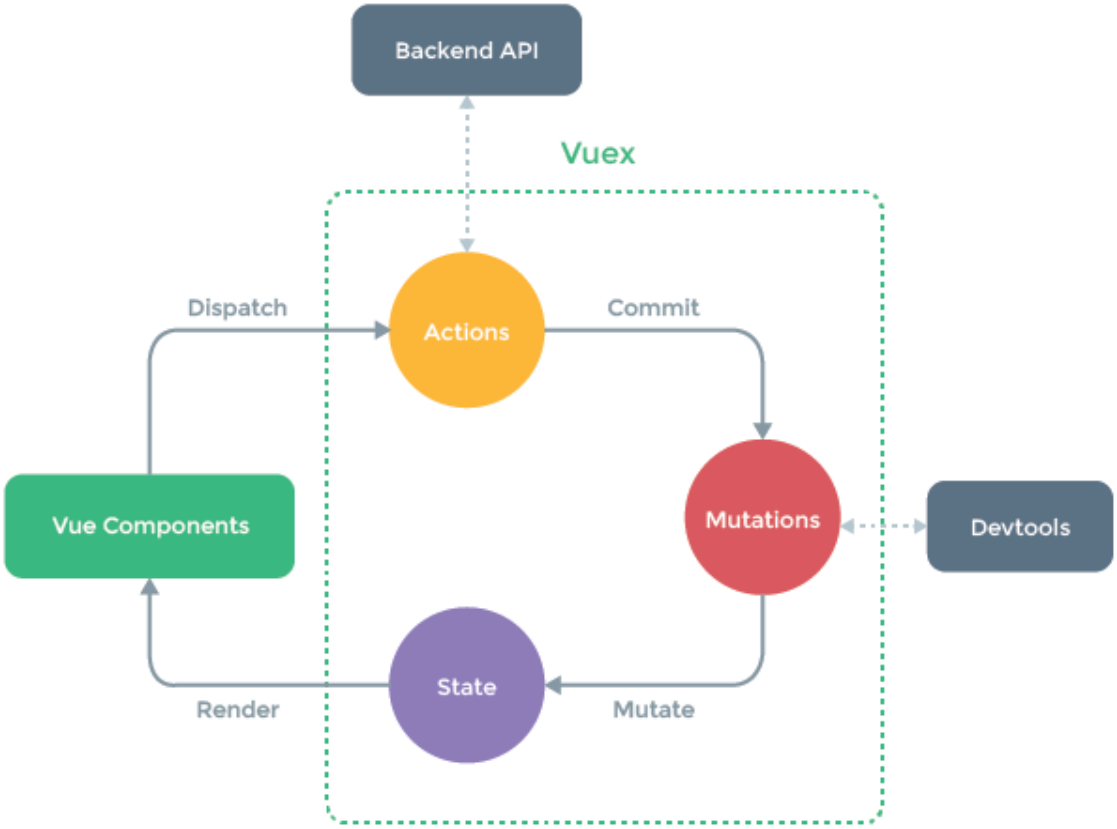
```
actions: {
    get_data_action(){
        axios.get('http://127.0.0.1:3000/data').then(res=>{

        });
    }
}
```

· 调用 **Actions** 的方法

```
this.$store.dispatch("方法名称",[payload])
```





2.WebStorage

Webstorage 提供了一种比 cookie 更加直观的数据存储方式，其以名/值对的形式进行存储。其分为两种存储机制：

sessionStorage,仅在浏览器窗口打开期间有效。

localStorage,其永远有效。

sessionStorage 和 localStorage 有相同的属性和方法

· length 属性

length 属性用于获取项目的数量，其语法结构是：

```
sessionStorage.length
```

```
localStorage.length
```

· setItem()方法

setItem()方法用于设置存储项目，其语法结构是：

```
sessionStorage.setItem(key,value)
```

```
localStorage.setItem(key,value)
```

· getItem()方法

getItem()用于获取项目的值，其语法结构是:

```
variable = sessionStorage.getItem(key)
```

```
variable = localStorage.getItem(key)
```

· removeItem()方法

removeItem()方法用于删除项目，其语法结构是:

```
sessionStorage.removeItem(key)
```

```
localStorage.removeItem(key)
```

· clear()方法

clear()方法用于删除所有项目，其语法结构是:

```
sessionStorage.clear()
localStorage.clear()
```

3.拖放

拖放即将源对象从一个位置拖动到另一个位置，在 HTML5 中任何元素都可以实现拖放，但为保证浏览器的兼容性，建议为被拖放的对象添加 draggable="true"属性。

3.1 拖放事件

· dragstart 事件

dragstart 事件在源对象开始被拖放时触发，其语法结构是：

```
HTMLElement.addEventListener('dragstart',
()=>{
  //...
});

HTMLElement.ondragstart = ()=>{
  //...
}
```

· drag 事件

drag 事件在源对象拖放过程中被触发，其语法结构是：

```
HTMLElement.addEventListener('drag',()=>{
  //...
});

HTMLElement.ondrag = ()=>{
  //...
}
```

· dragend 事件

dragend 事件在源对象释放后被触发(也可能在目标区域内，也可能在目标区域外)，其语法结构是：

```
HTMLElement.addEventListener('dragend',()
=>{
  //...
});
```

```
HTMLElement.ondragend = ()=>{
  //...
}
```

· dragenter 事件

dragenter 事件是在源对象进入目标对象的区域范围时被触发，其语法结构是：

```
HTMLElement.addEventListener('dragenter',
()=>{
  //...
});

HTMLElement.ondragenter = ()=>{
  //...
}
```

· dragover 事件

dragover 事件在源对象在目标对象悬停时触发，其语法结构是：

```
HTMLElement.addEventListener('dragover',
(event)=>{
    event.preventDefault();  //阻止默认事件的触发
  //...
});

HTMLElement.ondragover = (event)=>{
    event.preventDefault();
  //...
}
```

必须在该事件内调用 event.preventDefault()方法，否则不会触发 drop 事件

· dragleave 事件

dragleave 事件在源对象拖动离开目标对象时触发，其语法结构是：

```
HTMLElement.addEventListener('dragleave',
()=>{
  //...
});

HTMLElement.ondragleave = ()=>{
  //...
}
```

· drop 事件

drop 事件将在源对象在目标对象范围内释放时触发，其语法结构是：

```
HTMLElement.addEventListener('drop',()=>{
  //...
});
```

```
HTMLElement.ondrop = ()=>{
  //...
}
```

拖放事件的触发顺序:

dragstart --> drag --> dragenter-
->dragover--> drop-->dragend

3.2 dataTransfer 属性

拖放事件的 dataTransfer 属性将返回 DataTransfer 对象，用于保存拖放数据，其语法结构是:

```
DataTransfer DragEvent.dataTransfer
```

DragEvent 代表拖放事件对象

3.3 DataTransfer 对象

· setData()方法

setData()方法用于给指定的类型设置数据，如果类型不存在则自动添加到未尾，如果存在的话，则替换原来的值，其语法结构是：

DataTransfer.setData(type,value)　type 可以随便起名

· getData()方法

getData()方法用于获取指定类型的数据，其语法结构是：

variable = DataTransfer.getData(type)　type 可以随便起名

Day06:

HTML5 新特性 -- Unit06

1.Multer 中间件

1.1 概述

在实现文件上传时，表单必须存在以下属性：

· 表单的提交方式只能为 POST

· 必须设置表单的 enctype="multipart/form-data"属性

```
<form action="" method="post" enctype="multipart/form-data">
    ...
</form>
```

1.2 Multer 中间件

Multer 中间件用于处理存在<input type="file">类型的表单数据。

在通过 Multer 中间件进行文件上传时，上传文件的信息会自动存储到 req.file 或 req.files 属性中。

Multer 中间件在文件上传时：

A.为避免文件名称冲突，将自动进行重命名

B.文件上传时没有扩展名

安装 Multer

```
npm install --save multer
```

配置 Multer

```
//加载 Multer 中间件
const multer = require('multer');

//创建 Multer 实例
const upload = multer({
    //dest:destination,目标
    //指定文件上传的位置
    dest:'upload'
});
```

自定义存储规则 -- diskStorage()方法

```
Multer 实例.diskStorage({
    //用于定制上传目录的相关的规则
```

```
    //req 参数代表当前的HTTP 请求对象
    //file 参数代表当前上传的文件对象
    //cb(callback),回调函数,按指定的规则实现
上传
  destination:function(req,file,cb){

    },
    //用于定制上传文件名称的相关规则
    filename:function(req,file,cb){

    }
});
```

通过自定义规则创建 Multer 实例

```
let storage = multer.diskStorage({
  destination:function(req,file,cb){

  },
  filename:function(req,file,cb){
```

```
  }
});
const upload = multer({storage:storage});
```

Multer 实例的方法

· single()方法

single()方法用于实现单文件上传，其语法结构是：

```
Multer 实例.single('浏览框名称')
```

· array()方法

array()方法用于实现多文件上传，其语法结构是：

```
Multer 实例.array('浏览框名称')
```

上传文件信息：

originalname:上传的原始名称（包含扩展名称）

filename:上传后的名称

path:上传后的路径及名称

mimetype:上传文件 MIME 类型

destination:上传文件的位置

size:上传文件的尺寸(以字节为单位)

· UUID

UUID(Universally Unique Identifier)，通用唯一标识符，其目的是为了保证分布式系统中每一个元素都存在唯一的标识信息。

安装

```
npm install --save uuid
```

使用

```
const uuid = require('uuid');
//基于时间戳的 UUID
uuid.v1()

//基于随机数的 UUID
uuid.v4()
```

· DataTransfer 对象

DataTransfer 对象是通过拖放事件的 dataTransfer 属性返回的结果，其包含了拖放的数据。其包含了 setData()、getData()方法，还包含 files 属性。

files 属性将返回 FileList 对象，该对象既可以来源于表单中的 <input type="file">，也可能来源于用户的拖放操作，其语法结构是：

FileList DataTransfer 对象.files

FileList 对象存在 length 属性，用于获取包含的 <input type="file">或拖放文件的数量。

· FormData 对象

FormData 对象提供以键/值对表示的表单数据，经过它的数据可以用 AJAX 提交。

创建 FormData 对象：

```
variable = new FormData()
```

append()方法

append()方法用于添加一个新值到已有的键名上，如果键名不存在，则自动创建，其语法结构是：

```
FormData.append(key,value)
```

Day07:

HTML5 新特性 -- Unit07

1.SVG

1.1 什么是 SVG?

SVG(Scalable Vector Graphic)，可缩放的矢量图形。SVG 是基于 XML 的 2D 的图形格式。

1.2 SVG 的使用方式

· 直接通过<img>标签

示例代码如下：

```
<img src="svg/logo.svg" width="180" height
="80">
```

· 通过 CSS 中的 background 属性

示例代码如下:

```
#container{
    width: 900px;
    height: 550px;
    margin: 0 auto;
    border: 2px solid #f00;
    background: url(svg/logo.svg) no-repea
t center center;
}
```

· <object>标签

```
<object data="URL 地址" type="MIME 类型">
   浏览器不支持该类型文件时的提示信息
</object>
```

SVG 文件的 MIME 类型是: image/svg+xml

· <embed>标签

```
<embed src="URL 地址" type="MIME 类型" width
```

```
="宽度" height="高度">
    浏览器不支持该类型文件时的提示信息
</embed>
```

· **<iframe>标签**

```
<iframe src="URL 地址" width="宽度" height=
"高度" scrolling="是否显示滚动条(yes|no|aut
o)">
    浏览器不支持该类型文件时的提示信息
</iframe>
```

· **<svg>标签**

```
<svg version="1.1" xmlns="http://www.w3.or
g/2000/svg" width="宽度" height="高度">
    ...
</svg>
```

xmlns 是 XML Namespace 的缩写，意为 XML 命名空间，其根本作用是为了解决标签名称冲突。

1.3 svg 元素

· 元素属性

stroke-width 属性用于设置描边的宽度

stroke 属性用于设置描边的颜色

· **<line>元素**

<line>元素用于绘制线段，其语法结构是：

```
<line x1="起点 X 轴" y1="起点 Y 轴" x2="终点 X 轴" y2="终点 Y 轴"></line>
```

· **<polyline>元素**

<polyline>元素用于绘制折线，其语法结构是：

```
<polyline points="x1,y1,x2,y2,...">

</polyline>
```

· **<rect>元素**

<rect>元素用于绘制(圆角)矩形，其语法结构是：

```
<rect
      x="起点的 X 轴" y="起点的 Y 轴"
      width="宽度" height="高度"
      rx="rx" ry="ry">

</rect>
```

· <a>元素

<a>元素用于实现链接，其语法结构是：

```
<a
   xlink:href="目标文档 URL"
   xmlns:xlink="http://www.w3.org/1999/xlink">
    ...
</a>
```

http://www.zuohaotu.com/svg/

1.4 SVG DOM API

· document.createElementNS()方法

document.createElementNS()方法用于创建指定命名空间内的元素，其语法结构是：

```
Element document.createElementNS('命名空间','元素名称')
```

· 设置与获取元素属性

```
Element.setAttribute(name,value)
```

```
Element.getAttribute(name)
```

· 添加/删除子元素

```
Node Node.appendChild(subNode)
```

```
Node.removeChild(subNode)
```

2.ECharts

2.1 概述

ECharts 是百度推出的开源的数据可视化工具 --- 基于 Javascript 的图表库。

https://echarts.apache.org/zh/index.html

2.2 安装

· 浏览器

```
https://echarts.apache.org/zh/download.html
```

· npm

```
npm install --save echarts
```

2.3 基本使用

当在浏览器中引入外部的 JS 文件后，系统将自动暴露名称为 echarts 的对象

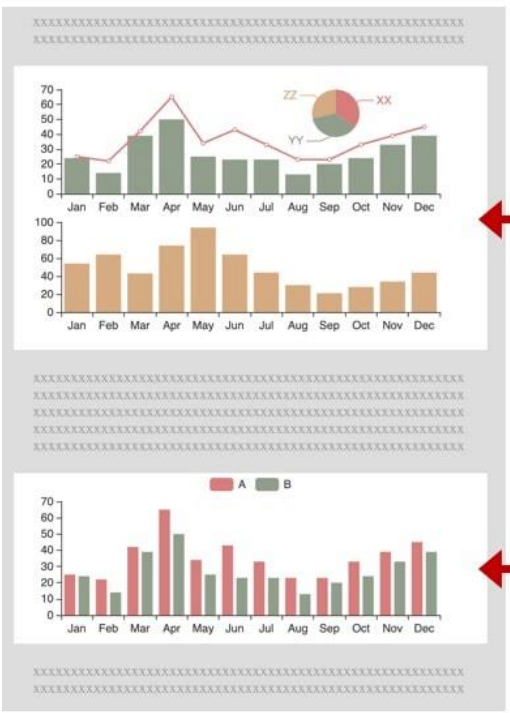
示例代码如下：

```
<script src="scripts/echarts.min.js"></script>
<script>
console.log(echarts);
</script>
```

2.4 基本术语

· 实例

在一个网页中可以存在多个实例，每个实例中可以有多个图表类型（如折线、柱形等）



instance 1

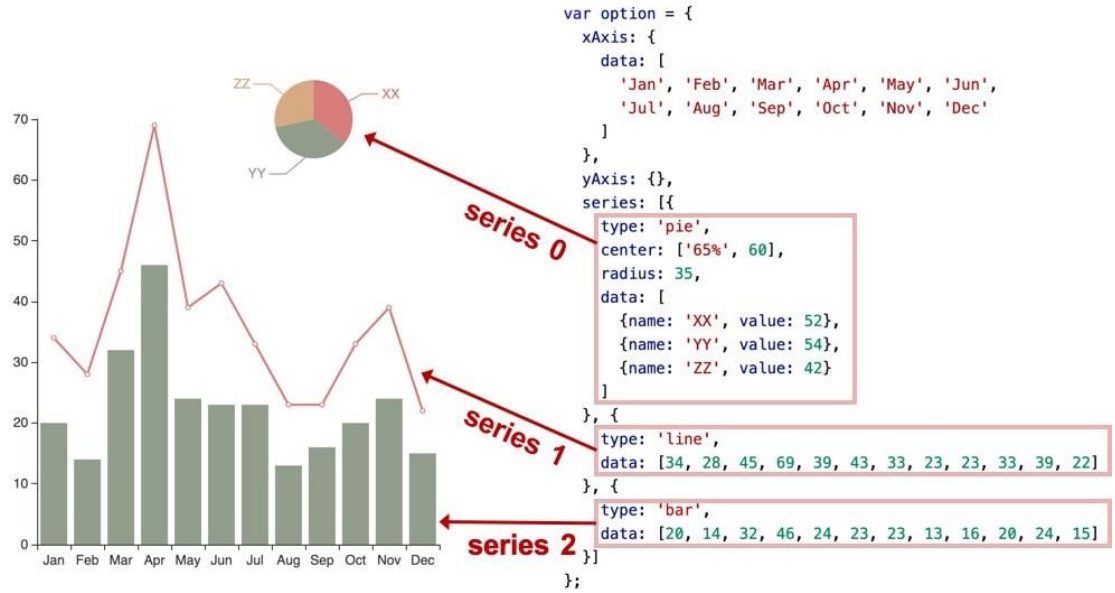
```
var dom1 = document.getElementById('id1');
var instance1 = echarts.init(dom1);
var option1 = {
  xAxis: [{...}, {...}],
  yAxis: [{...}, {...}],
  grid: [{...}, {...}],
  series: [
    {type: 'line', data: [...]},
    {type: 'bar', data: [...]},
    {type: 'bar', data: [...]},
    {type: 'pie', data: [...]}
  ]
};
instance1.setOption(option1);
```

instance 2

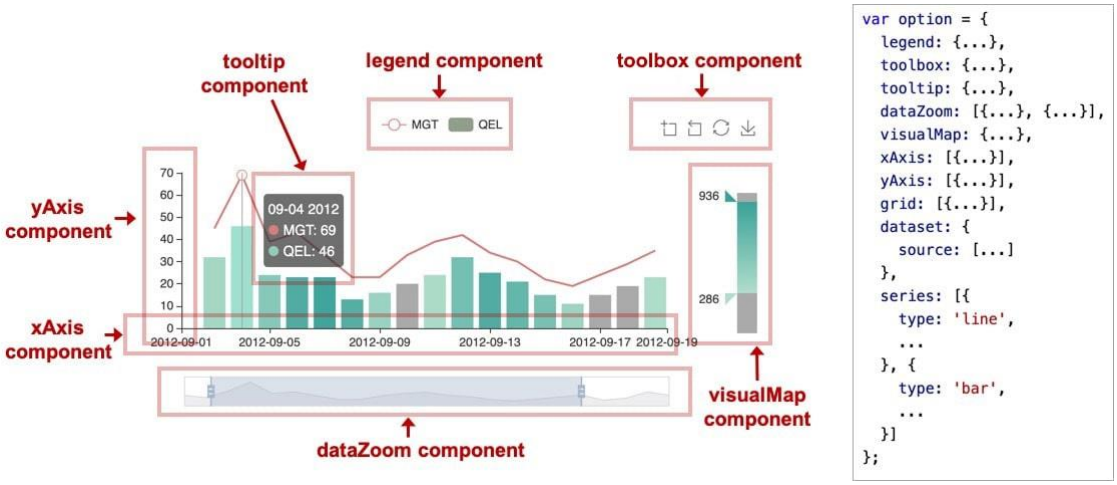
```
var dom2 = document.getElementById('id2');
var chart2 = echarts.init(dom2);
var option2 = {
  xAxis: {...},
  yAxis: {...},
  grid: {...},
  series: {type: 'bar', data: [...]}
};
chart2.setOption(option2);
```

· 系列

系列是实例中绘制的图表，一个实例中至少存在一个系列。



· 组件指图表的各个组件部分，如标题、X 轴、Y 轴等。



2.4 基本用法

A.在 HTML 页面中创建 DIV 元素，该 DIV 将作为图表渲染容器出现，必须为该 DIV 元素设置明确的宽度和高度

B.书写<script>标签，并且调用 echarts 对象的 init()方法以完成图表实例的创建，init()方法的语法结构建如下：

C.通过**图表实例**的 setOption()方法实现实例的配置

```
instance.setOption({...})
```

2.5 配置项

· title 组件

title 组件用于控制标题信息，其语法结构如下：

```
title:{
    show:是否显示标题信息(true|false),
```

```
    text:'控制主标题文本信息',
    link:'控制主标题的链接 URL 地址',
    target:'控制打开主标题链接的窗口形式(blank|self)',
    //控制主标题文本的样式
    textStyle:{
        color:'主标题文本颜色',
        fontSize:主标题文本字号(整数),
        fontFamily:'主标题文本字体',
        fontWeight:'主标题文本的加粗(normal|bold)',
        fontStyle:'主标题文本的倾斜(normal|italic)'
    },
    subtext:'副标题的文本信息',
    sublink:'控制副标题的链接 URL 地址',
    subtarget:'控制打开副标题链接的窗口形式(blank|self)',
    top:'title 组件离容器上侧的距离'(数字或字符串 top|middle|bottom),
    right:
    bottom:
```

left:'title 组件离容器左侧的距离'(数字或字符串 left|center|right)

}

xAxis

xAxis 属性用于控制 X 轴信息，其语法结构是：

```
xAxis:{
  show:是否显示 X 轴(true|false),
    type:'X 轴的类型(category|time)',
    data:该属性在 type 属性为 category 时必须存在,数组类型
}
```

yAxis

yAxis 属性用于控制 Y 轴信息，参见 X 轴

series

series 属性用于控制图表系列，其语法结构是：

```
series:[
  {
    type:'系列的名称(line|bar|pie)',
```

```
    data:系列的数据(数组·)
  },
  {
    type:'系列的名称(line|bar|pie)',
    data:系列的数据(数组·)
  }
]
```

A.注册一个全新的邮箱

B.下载微信小程序开发者工具

https://developers.weixin.qq.com/miniprogram/dev/devtools/download.html