Html5

# 新增标签

article: 标签定义外部的内容。

aside:标签定义 article 以外的内容。

audio:h5新增音频标签。没有高宽属性。

canvas:h5新增画布标签。

command: 定义命令按钮(未测试)

datalist：标签定义选项列表。

datalist 及其选项不会被想显示出来，它仅仅是合法的输入值列表。

details：标签用于描述文档或文档某个部分的细节。

figure：标签用于对元素进行组合。

figcaption：定义 figure 元素的标题。

footer：定义 section 或 document 的页脚。

header：定义 section 或 document 的页眉。

hgroup：用于对网页或区段（section）的标题进行组合。

keygen:标签规定用于表单的密钥对生成器字段

mark：标签定义带有记号的文本。

meter：通过min="0" max="20"的方式定义度量衡。仅用于已知最大和最小值的度量。

nav：定义document或section或article的导航。

output：定义不同的输出类型，比如脚本。

progress：定义任何类型的任务的进度。

rp:定义若浏览器不支持 ruby 元素显示的内容

rt：定义 ruby 注释的解释

ruby：定义 ruby 注释

section：标签定义文档中的节、区段。比如章节、页眉、页脚或文档中的其他部分。

source:audio和video的属性之一。为audio和video定义媒介源。

summary:为details定义标题。

time:定义日期或时间。

video：h5新增视频标签。具有高宽属性。

# Websocket

WebSocket 是HTML5提供的一种在单个 TCP 连接上进行全双工通讯的协议。WebSocket 使得客户端和服务器之间的数据交换变得更加简单，允许服务端主动向客户端推送数据。在 WebSocket API 中，浏览器和服务器只需要完成***一次握手***，两者之间就直接可以创建***持久性***的连接，自由进行数据传送和接收，并进行双向数据传输。不需要使用轮询的方式。

Ajax轮询。轮询是在特定的的时间间隔（如每1秒），由浏览器对服务器发出HTTP请求，然后由服务器返回最新的数据给客户端的浏览器。这种传统的模式带来很明显的缺点，即浏览器需要不断的向服务器发出请求，然而HTTP请求可能包含较长的头部，其中真正有效的数据可能只是很小的一部分，显然这样会浪费很多的带宽等资源。WebSocket能更好的节省服务器资源和带宽，并且能够更实时地进行通讯。

WebSocket 客户端基于事件的编程模型与 node 类似

通过 send() 方法来向服务器发送数据（send()方法只能接受字符串，json对象要先序列化成json字符串），close() 方法来关闭连接；

通过 onmessage 事件来接收服务器返回的数据，webSocket.onclose()连接关闭时会触发，.onopen()连接建立成功时触发。onerror()通信发生错误时触发

创建 WebSocket 对象：var Socket = new WebSocket(url, [protocol] );

第一个参数 url, 指定连接的 URL。第二个参数 protocol 是可选的，指定了可接受的子协议。

# Web存储localStorage和sessionStorage 以及cookie

HTML5 提供了两种在客户端存储数据的新方法：

* localStorage - 没有时间限制的数据存储
* sessionStorage - 针对一个 session 的数据存储

之前，这些都是由 cookie 完成的。但是 cookie 不适合大量数据的存储，因为它们由每个对服务器的请求来传递，这使得 cookie 速度很慢而且效率也不高。

在 HTML5 中，数据不是由每个服务器请求传递的，而是只有在请求时使用数据。它使在不影响网站性能的情况下存储大量数据成为可能。

对于不同的网站，数据存储于不同的区域，并且一个网站只能访问其自身的数据。

WebStorage不会传送到服务器，存储在本地的数据可以直接获取，也不会像cookie一样每次请求都会传送到服务器，所以减少了客户端和服务器端的交互，节省了网络流量；

安全性：WebStorage不会随着HTTP header发送到服务器端，所以安全性相对于cookie来说比较高一些，不会担心截获，但是仍然存在伪造问题；

localStorage 与 sessionStorage 的唯一一点区别就是 localStorage 属于永久性存储，而 sessionStorage 属于当会话结束的时候，sessionStorage 中的键值对会被清空。

| **特性** | **Cookie** | **localStorage** | **sessionStorage** |
| --- | --- | --- | --- |
| 数据的生命期 | 一般由服务器生成，可设置失效时间。如果在浏览器端生成Cookie，默认是关闭浏览器后失效 | 除非被清除，否则永久保存 | 仅在当前会话下有效，关闭页面或浏览器后被清除 |
| 存放数据大小 | 4K左右 | 一般为5MB | |
| 与服务器端通信 | 每次都会携带在HTTP头中，如果使用cookie保存过多数据会带来性能问题 | 仅在客户端（即浏览器）中保存，不参与和服务器的通信，不会自动把数据发送给服务器 | |
| 易用性 | 需要程序员自己封装，源生的Cookie接口不友好 | 源生接口可以接受，亦可再次封装来对Object和Array有更好的支持 | |
| 应用场景 | 登录、浏览记录 | 购物车、登录信息 | 敏感登录 |

## Cookie

http是一种无状态的协议，它的无状态可以用翻脸不认人（浏览器）来表示了；以至于服务器不会记得前一秒是哪个客户端向它发出了请求，即服务器不知道用户上一次做了什么，这严重阻碍了交互式Web应用程序的实现。所以使用cookie来解决。

cookie机制：如果不在浏览器中设置过期时间，cookie被保存在内存中，生命周期随浏览器的关闭而结束，这种cookie简称会话cookie。如果在浏览器中设置了cookie的过期时间，cookie被保存在硬盘中，关闭浏览器后，cookie数据仍然存在，直到过期时间结束才消失。

Cookie是服务器发给客户端的特殊信息，cookie是以文本的方式保存在客户端，每次请求时都带上它

存储内容：cookie只能保存字符串类型，以文本的方式

安全性：针对cookie所存在的攻击：Cookie欺骗，Cookie截获；session的安全性大于cookie。

Cookie的根本作用就是在客户端存储用户访问网站的一些信息。典型的应用有：

记住密码，下次自动登录。 购物车功能。

记录用户浏览数据，进行商品（广告）推荐。

缺陷：  
① Cookie会被附加在每个HTTP请求中，所以无形中增加了流量。  
② 由于在HTTP请求中的Cookie是明文传递的，所以安全性成问题。（除非用HTTPS）  
③ Cookie的大小限制在4KB左右。对于复杂的存储需求来说是不够用的。

## LocalStorage

localStorage 方法将数据保存在客户端本地的硬件设备，存储的数据没有时间限制。第二天、下一年之后，数据依然可用。

localStorage允许在浏览器中存储 key/value 对的数据。

localStorage 用于长久保存整个网站的数据，保存的数据没有过期时间，直到手动去删除。

localStorage 属性是只读的。

它将所有数据类型的数据都转换为字符串存储；

一个域名下的localStorage只能被同源网站读取（同源是指协议、主机名、端口号相同）。

localStorage的优势：

* 1、localStorage 拓展了 cookie 的 4K 限制。
* 2、localStorage 会可以将第一次请求的数据直接存储到本地，这个相当于一个 5M 大小的针对于前端页面的数据库，相比于 cookie 可以节约带宽，但是这个却是只有在高版本的浏览器中才支持的。

localStorage 的局限：

* 1、浏览器的大小不统一，并且在 IE8 以上的 IE 版本才支持 localStorage 这个属性。
* 2、目前所有的浏览器中都会把localStorage的值类型限定为string类型，这个在对我们日常比较常见的JSON对象类型需要一些转换。
* 3、localStorage在浏览器的隐私模式下面是不可读取的。
* 4、localStorage本质上是对字符串的读取，如果存储内容多的话会消耗内存空间，会导致页面变卡。
* 5、localStorage不能被爬虫抓取到。

语法： 获取方式：(window.localStorage)

localStorage ["key"]=value;

localStorage.key=value;

localStorage.setItem("key",value); //存

localStorage.getItem("key") //取

localStorage.removeItem("key"); //删除

localStorage.clear(); //删除所有数据

## sessionStorage

sessionStorage 将数据保存在session对象中。所谓session，是指用户在浏览某个网站时，从进入网站到浏览器关闭所经过的这段时间，也就是用户浏览这个网站所花费的时间。session对象可以用来保存在这段时间内所要求保存的任何数据。

sessionStorage 用于临时保存同一窗口(或标签页)的数据，在关闭窗口或标签页之后将会删除这些数据。 数据只保存在当前会话中

session机制：当服务器收到请求需要创建session对象时，首先会检查客户端请求中是否包含sessionid。如果有sessionid，服务器将根据该id返回对应session对象。如果客户端请求中没有sessionid，服务器会创建新的session对象，并把sessionid在本次响应中返回给客户端。通常使用cookie方式存储sessionid到客户端，在交互中浏览器按照规则将sessionid发送给服务器。如果用户禁用cookie，则要使用URL重写，可以通过response.encodeURL(url) 进行实现；API对encodeURL的结束为，当浏览器支持Cookie时，url不做任何处理；当浏览器不支持Cookie的时候，将会重写URL将SessionID拼接到访问地址后。

Cookie保存在客户端，session保存在服务器端，session的安全性大于cookie:

1、sessionID存储在cookie中，若要攻破session首先要攻破cookie；

2、sessionID是要有人登录，或者启动session\_start才会有，所以攻破cookie也不一定能得到sessionID；

3、第二次启动session\_start后，前一次的sessionID就是失效了，session过期后，sessionID也随之失效。

4、sessionID是加密的。

存储内容：cookie只能保存字符串类型，以文本的方式；session通过类似与Hashtable的数据结构来保存，能支持任何类型的对象(session中可含有多个对象).

sessionStorage允许在浏览器中存储 key/value 对的数据。

语法：

获取方式：window.sessionStorage // 返回一个存储对象

sessionStorage.setItem("key", "value"); //保存数据

var lastname = sessionStorage.getItem("key"); //读取数据

sessionStorage.removeItem("key"); //删除指定键的数据

sessionStorage.clear(); //删除所有数据

应用场景： 判断用户是否登录。 购物车功能。

# SVG

## 简介与优势

* SVG 指可伸缩矢量图形 (Scalable Vector Graphics)
* SVG 用来定义用于网络的基于矢量的图形
* SVG 使用 XML 格式定义图形
* SVG 图像在放大或改变尺寸的情况下其图形质量不会有所损失
* SVG 是万维网联盟的标准
* SVG 与诸如 DOM 和 XSL 之类的 W3C 标准是一个整体

**优势：**

* SVG 可被非常多的工具读取和修改（比如记事本）
* SVG 与 JPEG 和 GIF 图像比起来，尺寸更小，且可压缩性更强。
* SVG 是可伸缩的
* SVG 图像可在任何的分辨率下被高质量地打印
* SVG 可在图像质量不下降的情况下被放大
* SVG 图像中的文本是可选的，同时也是可搜索的（很适合制作地图）
* SVG 可以与 Java 技术一起运行
* SVG 是开放的标准
* SVG 文件是纯粹的 XML

SVG 的主要竞争者是 Flash。与 Flash 相比，SVG 最大的优势是与其他标准（比如 XSL 和 DOM）相兼容。而 Flash 则是未开源的私有技术。

## 元素与属性

SVG 有一些预定义的形状元素，可被开发者使用和操作：

矩形 <rect>

圆形 <circle>

椭圆 <ellipse>

线 <line>

折线 <polyline>

多边形 <polygon>

路径 <path>

<svg width="100%" height="100%" version="1.1"

xmlns="http://www.w3.org/2000/svg">

<circle cx="100" cy="50" r="40" stroke="black"

stroke-width="2" fill="red"/>

</svg>

SVG 代码以 <svg> 元素开始，包括开启标签 <svg> 和关闭标签 </svg> 。这是根元素。关闭标签的作用是关闭 SVG 元素和文档本身。

width 和 height 属性可设置此 SVG 文档的宽度和高度。version 属性可定义所使用的 SVG 版本，xmlns 属性可定义 SVG 命名空间。

stroke 属性设置轮廓颜色， stroke-width 属性设置轮廓宽度。我们把圆的轮廓设置为 2px 宽，黑边框。

fill 属性设置形状填充的颜色。我们把填充颜色设置为红色。fill-opacity 属性定义填充颜色透明度（合法的范围是：0 - 1），stroke-opacity 属性定义轮廓颜色的透明度（合法的范围是：0 - 1），opacity 属性定义整个元素的透明值（合法的范围是：0 - 1）

<circle> 用来创建一个圆。cx 和 cy 属性定义圆中心的 x 和 y 坐标。如果忽略这两个属性，那么圆点会被设置为 (0, 0)。r 属性定义圆的半径。

矩形中：rx 和 ry 属性可使矩形产生圆角。

椭圆中：rx 属性定义水平半径；ry 属性定义垂直半径

线条中：x1 属性在 x 轴定义线条的开始；y1 属性在 y 轴定义线条的开始

x2 属性在 x 轴定义线条的结束；y2 属性在 y 轴定义线条的结束

多边形中：points 属性定义多边形每个角的 x 和 y 坐标

折线中：points 属性定义每条线转折点的 x 和 y 坐标

# Canvas画布

HTML5中 <canvas> 标签用于绘制图像，<canvas> 元素本身并没有绘制能力（它仅仅是图形的容器） - 必须使用脚本来完成实际的绘图任务。

画布是一个矩形区域，可以控制其每一像素。它拥有多种绘制路径、矩形、圆形、字符以及添加图像的方法。

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。

getContext("2d") 对象是内建的 HTML5 对象，拥有多种绘制路径、矩形、圆形、字符以及添加图像的方法。

## 用法：

var canvas=document.getElementById("myCanvas");

var content=canvas.getContext("2d");

### 画矩形：

1-> 空心 strokeRect() 默认为黑色 2 ->实心 fillRect() fillStyle=’color’

语法：context.strokeRect(x起始x坐标,y起始y坐标,width,height)

context.clearRect(x,y,width,height)：在给定矩形内清空一个矩形。

### 画线条：

context.moveTo(x,y)  线条开始坐标 context.lineTo(x,y)  线条结束坐标

beginPath():开始一条路径 colsePath():关闭一条路径（从终点到起点）

context.stroke() 绘制已定义的路径/轮廓（如果未调用则不会画线）

context.strokeStyle="color" 定义线条颜色

context.fillStyle="color" context.fill() 实心填充线条内部颜色

### 画圆：

arc() 方法创建弧/曲线（用于创建圆或部分圆）。

如需通过 arc() 来创建圆，请把起始角设置为 0，结束角设置为 2\*Math.PI。

context.arc(x,y,r,sAngle,eAngle,counterclockwise)方法：

|  |  |
| --- | --- |
| *x* | 圆的中心的 x 坐标。 |
| *y* | 圆的中心的 y 坐标。 |
| *r* | 圆的半径。 |
| *sAngle* | 起始角，以弧度计。（弧的圆形的三点钟位置是 0 度）。 |
| *eAngle* | 结束角，以弧度计。（1\*Math.PI为半圆，类推） |
| *counterclockwise* | 可选。规定是逆时针or顺时针绘图。（默认）False = 顺时针，true = 逆时针。 |

### 绘制文本：

先定义字体：context.font="30px Arial";

.font设置或返回文本内容的当前字体属性,与css的font属性相同，可设置所有字体属性

.textAlign=”start” 文本对齐方式：

start默认:文本在指定的位置开始。 end 文本在指定的位置结束。

center:文本的中心被放置在指定的位置。 left:文本左对齐。right 文本右对齐。

fillText(*text,x,y*) - 在 canvas 上绘制实心的文本 .fillStyle="color"设置颜色

strokeText(*text,x,y*) - 在 canvas 上绘制空心的文本

### 渐变：

线性渐变：createLinearGradient(x1,y1,x2,y2)     创建线性的渐变对象。

（x1,y1）是渐变开始的坐标；

（x2,y2）是渐变结束的坐标。

放射性渐变：createRadialGradient(x1,y1,r1,x2,y2,r2) 创建放射状/圆形渐变对象。

（x1,y1）渐变开始的圆的坐标 r1：渐变开始的圆的半径

（x2,y2）渐变结束的圆的坐标 r1：渐变结束的圆的半径

addColorStop(x,color) 方法规定在x处的颜色，即在 gradient 对象中的何处定位颜色。

添加渐变点：addcolorStop(位置，颜色)

例：addcolorStop(0,"pink")    0是开始，1是结束。

context.fillStyle=grd; //将gradient 对象赋给填充样式或轮廓样式strokeStyle

context.fillRect(0,0,150,100); //规定以什么形式进行渐变，可为矩形圆形线条文本等

### 绘制图片：

等图片加载完在执行canvas操作，所以在onlcoad中调用方法

先创建image（）图像 drawImage() 方法在画布上绘制图像、画布或视频。

语法1：context.drawImage(img,x,y)

语法2：context.drawImage(img,x,y,width,height)规定图像的宽度和高度

语法3：context.drawImage(img,sx,sy,swidth,sheight,x,y,width,height)剪切图像，并定位被剪切的部分

|  |  |
| --- | --- |
| *img* | 规定要使用的图像、画布或视频。 |
| *sx* | 可选。开始剪切的 x 坐标位置。 |
| *sy* | 可选。开始剪切的 y 坐标位置。 |
| *swidth* | 可选。被剪切图像的宽度。 |
| *sheight* | 可选。被剪切图像的高度。 |
| *x* | 在画布上放置图像的 x 坐标位置。 |
| *y* | 在画布上放置图像的 y 坐标位置。 |
| *width* | 可选。要使用的图像的宽度。（伸展或缩小图像） |
| *height* | 可选。要使用的图像的高度。（伸展或缩小图像） |

# Canvas与SVG的比较

SVG 是一种使用 XML 描述 2D 图形的语言。基于 XML，这意味着 SVG DOM 中的每个元素都是可用的。您可以为某个元素附加 JavaScript 事件处理器。

在 SVG 中，每个被绘制的图形均被视为对象。如果 SVG 对象的属性发生变化，那么浏览器能够自动重现图形。

Canvas 通过 JavaScript 来绘制 2D 图形。是逐像素进行渲染的。

在 canvas 中，一旦图形被绘制完成，它就不会继续得到浏览器的关注。如果其位置发生变化，那么整个场景也需要重新绘制，包括任何或许已被图形覆盖的对象。

**Canvas**

* 依赖分辨率
* 不支持事件处理器
* 弱的文本渲染能力
* 能够以 .png 或 .jpg 格式保存结果图像
* 最适合图像密集型的游戏，其中的许多对象会被频繁重绘
* Canvas提供的功能更原始，适合像素处理，动态渲染和大数据量绘制

**SVG**

* 不依赖分辨率
* 支持事件处理器
* 最适合带有大型渲染区域的应用程序（比如谷歌地图）
* 复杂度高会减慢渲染速度（任何过度使用 DOM 的应用都不快）
* 不适合游戏应用
* SVG功能更完善，适合静态图片展示，高保真文档查看和打印的应用场景；

# HTML记忆知识：

## Browser 对象

history Location navigator screen

<meta> 元素可提供有关页面的元信息（meta-information），比如针对搜索引擎和更新频度的描述和关键词。

<meta> 标签位于文档的头部，不包含任何内容。

<meta> 标签的属性定义了与文档相关联的名称/值对。

<mark> 标签定义带有记号的文本。请在需要突出显示文本时使用 <m> 标签。

# CSS相关

## 新增属性：

1、box-shadow（阴影效果）

2、border-color（为边框设置多种颜色）

3、border-image（图片边框）

4、text-shadow（文本阴影）

5、text-overflow（文本截断）

6、word-wrap（自动换行）

7、border-radius（圆角边框）

8、opacity（透明度）

9、box-sizing（控制盒模型的组成模式）

10、resize（元素缩放）

11、outline（外边框）

12、background-size（指定背景图片尺寸）

13、background-origin（指定背景图片从哪里开始显示）

14、background-clip（指定背景图片从什么位置开始裁剪）

15、background（为一个元素指定多个背景）

16、hsl（通过色调、饱和度、亮度来指定颜色颜色值）

17、hsla（在hsl的基础上增加透明度设置）

18、rgba（基于rgb设置颜色，a设置透明度）

## transform属性：

rotateY：定义3D 旋转，rotateY可实现翻书的效果

translate：移动元素

scale：定义 2D 缩放转换时

perspective(n)：改变3D元素是怎样查看透视图（注：只影响 3D 转换元素）

## 显示与隐藏属性：

display：none指的是元素完全不陈列出来，不占据空间，涉及到了DOM结构，故产生reflow回流与repaint重绘

visibility：hidden指的是元素不可见但存在，保留空间，不影响结构，故只产生repaint，但不可触发绑定事件

opacity=0：指的是元素不可见但存在，保留空间，不影响结构，并且，如果该元素已经绑定一些事件，如click事件，那么点击该区域，也能触发

# 页面性能指标：

白屏时间（first Paint Time）——用户从打开页面开始到页面开始有东西呈现为止

首屏时间——用户浏览器首屏内所有内容都呈现出来所花费的时间

用户可操作时间(dom Interactive)——用户可以进行正常的点击、输入等操作，默认可以统计domready时间，因为通常会在这时候绑定事件操作

总下载时间——页面所有资源都加载完成并呈现出来所花的时间，即页面 onload 的时间

总加载时间 = 白屏时间(first paint) + 首屏时间 + 核心功能可用时间 + 页面可交互(DOM Ready).

# 页面加载顺序：

Readystatechange:  
document有readyState属性来描述document的loading状态，readyState的改变会触发readystatechange事件.

* + loading  
    文档仍然在加载
  + interactive  
    文档结束加载并且被解析，但是像图片，样式，frame之类的子资源仍在加载.

此时readyState为'interactive'，表明document已经load并解析完成，触发 readystatechange，然后触发DOMContentLoaded(在大多数浏览器上的表现如此)。捎带提一句，此时，加载完成且带有defer标记的脚本，会按顺序开始执行。

* + complete  
    文档和子资源已经结束加载，该状态表明将要触发load事件。

因此readystatechange在onload之前触发。

4、DOMContentLoaded：  
当初始的 HTML 文档被完全加载和解析完成之后，DOMContentLoaded 事件被触发，而无需等待样式表、图像和子框架的完成加载。  
另一个不同的事件 load 应该仅用于**检测一个完全加载的页面**。因此DOMContentLoaded是HTML完全加载和解析完成之后发生的，发生时间点要早于load，   
在使用 DOMContentLoaded 更加合适的情况下使用 load 是一个令人难以置信的流行的错误，所以要谨慎。  
注意：DOMContentLoaded 事件必须等待其所属script之前的样式表加载解析完成才会触发。

5、Onpageshow：  
onpageshow 事件在用户浏览网页时触发。  
onpageshow 事件类似于 onload 事件，onload 事件在页面第一次加载时触发， onpageshow 事件在每次加载页面时触发，即 onload 事件在页面从浏览器缓存中读取时不触发。

6、Beforeunload：  
当浏览器窗口，文档或其资源将要卸载时，会触发beforeunload事件。这个文档是依然可见的，并且这个事件在这一刻是可以取消的.  
如果处理函数为Event对象的returnValue属性赋值非空字符串，浏览器会弹出一个对话框，来询问用户是否确定要离开当前页面（如下示例）。有些浏览器会将返回的字符串展示在弹框里，但有些其他浏览器只展示它们自定义的信息。没有赋值时，该事件不做任何响应。

# JS相关：

stopPropagation() 阻止事件冒泡。 这个事件不会阻止定义在元素上的其他事件。

stopImmediatePropagation() 会彻底的阻止事件， 在其之后的绑定在元素上的其他监听事件都不会触发

preventDefault() 阻止事件的默认动作

# 前端性能优化：

在渲染DOM的时候，浏览器所做的工作实际上是：

1. 获取DOM后分割为多个图层  
   2. 对每个图层的节点计算样式结果（Recalculate style--样式重计算）  
   3. 为每个节点生成图形和位置（Layout--回流和重布局）  
   4. 将每个节点绘制填充到图层位图中（Paint Setup和Paint--重绘）  
   5. 图层作为纹理上传至GPU  
   6. 符合多个图层到页面上生成最终屏幕图像（Composite Layers--图层重组）

Chrome中满足以下任意情况就会创建图层：  
\* 3D或透视变换（perspective transform）CSS属性  
\* 使用加速视频解码的<video>节点  
\* 拥有3D（WebGL）上下文或加速的2D上下文的<canvas>节点  
\* 混合插件（如Flash）  
\* 对自己的opacity做CSS动画或使用一个动画webkit变换的元素  
\* 拥有加速CSS过滤器的元素  
\* 元素有一个包含复合层的后代节点（一个元素拥有一个子元素，该子元素在自己的层里）  
\* 元素有一个z-index较低且包含一个复合层的兄弟元素（换句话说就是该元素在复合层上面渲染）

**注意❗：**如果图层中某个元素需要重绘，那么整个图层都需要重绘。比如一个图层包含很多节点，其中有个gif图，gif图的每一帧，都会重回整个图层的其他节点，然后生成最终的图层位图。所以这需要通过特殊的方式来强制gif图属于自己一个图层（translateZ(0)或者translate3d(0,0,0)），CSS3的动画也是一样（好在绝大部分情况浏览器自己会为CSS3动画的节点创建图层）。

## 重排和重绘：(可参考html文档)

浏览器下载完页面中的所有组件——HTML标记、JavaScript、CSS、图片之后会解析生成两个内部数据结构——DOM树和渲染树（在构建渲染树之前，和DOM树同期会构建Style Tree。DOM树与Style Tree合并为渲染树）

DOM树表示页面结构，渲染树表示DOM节点如何显示。

一旦DOM和渲染树构建完成，浏览器就开始显示（绘制）页面元素。  
当DOM的变化影响了元素的几何属性（比如宽、高或位置），导致浏览器需要重新计算元素的几何属性，其他元素的几何属性和位置也会因此受到影响。浏览器会使渲染树中受到影响的部分失效，并重新构造渲染树。**这个过程称为重排**reflow，重排负责元素的几何属性更新。完成重排后，浏览器会重新绘制受影响的部分到屏幕，**该过程称为重绘**repaint。重绘负责元素的样式更新。

重排一定会导致重绘，重绘不一定会发生重排。

table及其内部元素除外，它可能需要多次计算才能确定好其在渲染树中节点的属性，通常要花3倍于同等元素的时间。这也是为什么要避免使用table做布局的一个原因。

### 重排触发机制：

1、添加或者删除可见的DOM元素  
2、元素位置改变  
3、元素尺寸改变  
4、元素内容改变（例如：一个文本被另一个不同尺寸的图片替代）  
5、页面渲染初始化（这个无法避免）  
6、浏览器窗口尺寸大小发生改变

触发属性：

div盒子模型：Width height padding margin display border-width border min-height

定位属性和float：top bottom left right position float clear

节点内部文字结构：text-align overflow-y font-weight overflow font-family

line-height vertival-align white-space font-size

重绘：元素的属性或者样式发生变化。

触发属性：color border-style border-radius visibility text-decoration  
background background-image background-position background-repeat  
background-size outline-color outline outline-style outline-width

box-shadow

透明度（opacity）不会触发重绘：实际上透明度的改变后，GPU在绘画时只是简单的降低之前已经画好的纹理的alpha值来达到效果，并不需要整体的重绘。不过这个前提是这个被修改opacity本身必须是一个图层，如果图层下还有其他节点，GPU也会将他们透明化。

div.style.width = '200px'; div.style.background = 'red';

div.style.height = '300px'; //久远浏览器时，分别设置宽高时，会触发两次重排

大多数浏览器通过队列化修改并批量执行来优化重排过程。即以上代码只会构成一次重排。

但是还是有一些特殊的元素几何属性会造成这种优化失效。

比如：offsetTop, offsetLeft,scrollTop, scrollLeft, clientTop, clientLeft, 请求了getComputedStyle()或者 IE的 currentStyle。以上都需要实时回馈给用户的几何属性或者是布局属性，不能再依靠浏览器的优化，因此浏览器不得不立即执行渲染队列中的“待处理变化”，并随之触发重排返回正确的值。

### **避免重绘和重排：**

将需要多次重排的元素，position属性设为absolute或fixed，这样此元素就脱离了文档流，它的变化不会影响到其他元素。例如有动画效果的元素就最好设置为绝对定位。

改变样式时：

1、使用内联样式的cssText方法：el.style.cssText = 'border-left: 1px; border-right: 2px; padding: 5px';

2、切换类名：el.className = 'active';

批量修改DOM：（以下过程引发两次重排：1和3）（fragment元素的应用）

1、让该元素脱离文档流 2、对其进行多重改变 3、将元素带回文档中

(方法：改变display、fragment文档碎片(最推荐)、克隆节点)

缓存布局信息：

为了尽量减少对布局信息的查询次数，查询时，将其赋值给局部变量，使用局部变量参与计算。可以避免重复取值访问DOM

强迫浏览器创建图层：（使用3d硬件，不应过多）

在Blink和WebKit的浏览器中，一当一个节点被设定了透明度的相关过渡效果或动画时，浏览器会将其作为一个单独的图层，但很多开发者使用translateZ(0)或者translate3d(0,0,0)去使浏览器创建图层。这种方式可以消除在动画开始之前的图层创建时间，使得动画尽快开始（创建图层和绘制图层还是比较慢的），而且不会随着抗锯齿而导出突变。不过这种方法需要节制，否则会因为创建过多的图层导致崩溃

Chrome中的抗锯齿：

Chrome中，非根图层以及透明图层使用grayscale antialiasing而不是subpixel antialiasing，如果抗锯齿方法变化，这个效果将会非常显著。如果你打算预处理一个节点而不打算等到动画开始，可以通过这种强迫浏览器创建图层的方式进行

修改节点位置时：

使用left和top会触发重布局，修改时的代价相当大。取而代之的更好方法是使用translate，这个不会触发重布局

## JS动画和CSS3动画

避免使用动画会触发重布局和重绘的属性，以免失帧，提前申明动画，这样能让浏览器提前对动画进行优化。由于GPU的参与，现在用来做动画的最好属性是如下几个：  
opacity translate rotate scale

### JS动画

缺点：JavaScript在浏览器的主线程中运行，而其中还有很多其他需要运行的JavaScript、样式计算、布局、绘制等对其干扰。这也就导致了线程可能出现阻塞，从而造成丢帧的情况。

优点：JavaScript的动画与CSS预先定义好的动画不同，可以在其动画过程中对其进行控制：开始、暂停、回放、中止、取消都是可以做到的。而且一些动画效果，比如视差滚动效果，只有JavaScript能够完成

### CSS3动画

缺点：缺乏强大的控制能力。而且很难以有意义的方式结合到一起，使得动画变得复杂且易于出问题。  
优点：浏览器可以对动画进行优化。它必要时可以创建图层，然后在主线程之外运行。

# HTTP

**Request Headers 请求头**

1. Accept:告诉服务器，客户机支持的数据类型
2. Accept-Encoding:告诉服务器，客户机支持的数据压缩格式
3. Cache-Control：缓存控制，服务器通过控制浏览器要不要缓存数据
4. Connection:处理完这次请求，是断开连接还是保持连接
5. Cookie:客户机通过这个可以向服务器带数据
6. Host:访问的主机名
7. Upgrade-Insecure-Requests
8. User-Agent:告诉服务器，客户机的软件环境

**Response Headers响应头**

1. Connection:处理完这次请求后，是断开连接还是继续保持连接
2. Content-Encoding:服务器通过这个头告诉浏览器数据的压缩格式
3. Content-Length:服务器通过这个头告诉浏览器回送数据的长度
4. Content-Type:服务器通过这个头告诉浏览器回送数据的类型
5. Date:当前时间值
6. Server:服务器通过这个头告诉浏览器服务器的类型
7. Vary:Accept-Encoding ——明确告知缓存服务器按照 Accept-Encoding 字段的内容，分别缓存不同的版本;
8. X-Powered-By:服务器告知客户机网站是用何种语言或框架编写的。