# 浏览器

## 16ms渲染帧

屏幕刷新率60HZ，1秒刷新60次，则1000/60=16ms

## 重绘和重排（回流）

* 重排：是改变DOM元素的位置和大小（即改变了几何属性）时，会导致浏览器重新生成渲染树，这个过程叫重排。
* 重绘：当重新生成渲染树后，就要将渲染树每个节点绘制到屏幕，这个过程叫重绘。

不是所有的动作都会导致重排，比如改变字体颜色，只会导致重绘。

重排会导致重绘，重绘不会导致重排

重绘和重排这两个操作都是非常昂贵的，因为JavaScript引擎线程和GUI渲染线程是互斥的，他们同时只能一个在工作。

**如何减少重排和重绘？**

1. 用JavaScript修改样式时，最好不要直接写样式，而是替换class来改变样式
2. 如果要将DOM元素执行一系列操作，可以将DOM元素脱离文档流，修改完成后，再将它带回文档。

推荐使用隐藏元素（display：none）或文档碎片（DocumentFragement），都能很好的实现这个方案。

1. 将动画效果应用到position属性为absolute [ˈæbsəluːt]和fixed的元素上；
2. 使用resize、scroll等进行防抖/节流处理；
3. 避免频繁读取对引发回流和重绘的属性，如果确实要多次使用，就用一个变量缓存起来；
4. 利用css3的transform、opacity、filter这些属性可以实现合成的效果，也就是GPU加速。

提升为合成层简单来说有以下好处：

* 合成层的位图，会交由GPU合成，比CPU处理要快；
* 当需要重绘时，只需要重绘本身，不会影响到其他的层；

1. 对于Render Tree的计算通常只需要遍历一次就可以完成，但table及其内部元素除外，他们可能需要多次计算，通常需要花3倍于同等元素的时间，这也是为什么要避免使用table布局的原因之一。

## 进程和线程

多线程可以并行处理任务，但是线程不能单独存在，它是有进程来启动和管理的；

一个进程就是一个程序的运行实例。启动程序的时候，操作系统会为该程序创建一块内存，用来存放代码，运行中的数据和一个执行任务的主线程，我们把这样的一个环境叫线程。

线程是依附于进程的，而进程中使用多线程并行处理能提升运算效率。

进程和线程之间的关系有如下4个特点：

1. 进程中的任意一线程执行出错，都会导致整个进程的崩溃；
2. 线程之间共享进程中的数据；
3. 当一个进程关闭之后，操作系统会回收进程所占用的内存
4. 进程之间的内容相互隔离

## 浏览器渲染机制

**9.1 浏览器是多进程的，包含哪些进程？**

* 浏览器主线程
* 渲染进程
* GPU进程
* 网络进程
* 插件进程

**9.2 浏览器的渲染进程是多线程的，包含哪些线程？**

* **GUI渲染线程**
* **JS引擎线程**
* **事件触发线程**
* **定时触发器线程**
* **异步http请求线程**

**9.3 浏览器的渲染流程**

## **渲染引擎和JavaScript引擎是怎么通信的**

## **常见的浏览器兼容性问题**

1. **不同浏览器的默认margin和padding不一样**
2. **图片的默认间距不一样**
3. **获取视口的宽高window.innerHeight/width**
4. **css的动画、过渡、渐变、flex也有，grid**
5. **Canvas、SVG**
6. **IE9以下不能的opacity，使用filter：alpha（opacity = 50）**
7. **event.offsetX/Y**
8. **绑定事件（addEventListener）IE9才支持**

## 不受同源限制的资源

网页不能请求不同源服务器上的资源，也不能读取不同源网页的cookie、localStorage中的数据，但是以下资源的请求不受同源策略的限制：

* <script src=”…”></script>
* <link rel=”stylesheet” href=”…”>
* <img src=”…” />
* <video>和<audio>嵌入的多媒体资源
* <object>、<embed>和<applet>的插件
* @font-face引入的字体。一般浏览器允许跨域字体(cross-origin fonts),一些需要同源字体(same-origin fonts)
* <frame>和<iframe>载入的任何资源(可以使用X-Frame-Options阻止这种形式的跨域交互)

我们平常说的最多的跨域受限的方式是采用ajax的显示请求数据。

1. 而CRSF可以利用上叙不受跨域资源影响的标签对目标服务器进行攻击。

## **web安全XSS、CSRF**

1. **xss：跨站脚本攻击**

* xss的重点不在于跨站点，而在于脚本的执行
* 原理：恶意攻击者在web页面中会插入一些恶意的script代码；

当用户浏览该页面的时候，那么嵌入在web页面的script代码会执行；

以此达到恶意攻击用户的目的。

* 分类：

（1）反射型XSS：在url后面的参数中加入恶意攻击代码

（2）存储型XSS：将恶意代码存储在服务器上

防范：后端需要对提交的数据进行过滤

（3）DOM-based型XSS：客户端的js对页面dom节点进行动态操作

* 防范：

（1）cookie安全策略

* http-only：只允许http或https请求读取cookie，js代码是无法读取cookie的；

设置方式：在响应头Set-Cookie：http-only

* secure：只允许https请求读取 [sɪˈkjʊə(r)]

设置方式：在响应头Set-Cookie：secure

* host-only：只允许主机域名和domain设置完成一致的网站才能访问该cookie

（2）防御HTML、HTML Attribute、javascript、URL、CSS编码

1. **CSRF：跨域请求伪造**

* 原理：用户登录一个正常的网站后，由于没有退出该正常网站，cookie信息还保留，然后用户去点击一个危险的网站页面，那么这个时候危险网站就可以拿到你之前登录的cookie信息，然后使用cookie信息去做其他事情。
* CSRF攻击并没有违反同源策略
* 防范：

（1）验证HTTP Referer字段：判断来源域名是否是本地网站

（2）加验证码

（3）使用Token

（4）sameSite

Cookie的SameSite属性用来限制第三方Cookie，从而减少风险。

设置方式：在响应头Set-Cookie：SameSite=Lax

它可以设置三个值:

* Strict 最严格，完全禁止第三方Cookie，跨站点时，任何情况下都不会发送Cookie，换言之，只有当前页面的URL与全球目标一致，才会带上Cookie。
* Lax 大多数情况是不发送第三方Cookie，但是导航到目标网站的Get请求除外。
* None Chrome计划将Lax变为默认设置，显式关闭SameSite属性，将其设为None。不过，前提是必须同时设置Secure属性，否则无效。

## HTTPS SSL 中间人劫持

SSL劫持攻击即SSL证书欺骗攻击，攻击者为了获得HTTPS传输的明文信息，需要先将自己接入到客户端和服务端之间，在传输过程中伪造服务器的证书，将服务器的公匙替换成自己的公匙，这样，中间人就可以得到明文传输的客户端随机数1、服务端随机数2、客户端公匙加密的随机数3、从而窃取客户端和服务端的通信数据。

但是对于客户端浏览器来说，如果中间人伪造了证书，在校验证书过程中会提示证书错误，由用户选择继续操作还是返回，由于大多数用户的安全意识不强，会选择继续操作，此时，中间人就可以获取浏览器和服务器之间的通信数据。

## HOST、origin、referer的区别

## 一些安全相关的HTTP响应头

现代浏览器提供了一些安全相关的响应头，使用这些响应头一般只需要修改服务器配置即可，不需要修改程序代码，成本很低

1. **Strict Transport Security**

* HTTP Strict Transport Security，简称为HSTS。它允许一个HTTPS网站，要求服务器总是通过HTTPS来访问。
* 当用户主动输入http请求时，一般我们会通过Web Server发送301/302重定向来解决这个问题。
* 现在有了HSTS，可以让浏览器帮你做这个跳转，省一次HTTP请求。
* 如何使用?

strict-transport-security: max-age=16070400; includeSubDomains

1. includeSubDomains是可选的，用来指定是否作用于子域名。

* 支持HSTS的浏览器遇到这个响应头，会把当前网站加入HSTS列表，然后在max-age指定的秒数内，当前网站所有的请求会被重定向为https。
* Chrome可以通过输入chrome://net-internals/#hsts查看。

**2、X-Frame-Options**

* X-Frame-Options，是为了减少点击劫持而引入的响应头；
* 使用方式：

x-frame-options：SAMEORIGIN

这个响应头支持三种配置

1. DENY：不允许任何页面嵌入；
2. SAMEORIGIN：不允许被本域以外的页面嵌入
3. Allow-FrOM uri：不允许被指定的域名以外的页面 嵌入

* 如果某页面被不允许的页面以<iframe>或<frame>的形式嵌入，IE会显示类似于“此内容无法在框架中显示”的提示信息，chrome和Firefox都会在控制台打印信息。
* 由于嵌入的页面不会加载，这就减少了点击劫持的发送。

**3、X-XSS-Protection**

顾名思义，这个响应头是用来防范XSS的。默认都开启，使用这个header可以关闭它。它的几种配置：

0 : 禁用XSS保护

1 ： 启用XSS保护

1;mode=block : 启用XSS保护，并在检查到XSS攻击时，停止渲染页面。

浏览器提供的XSS保护机制并不完美，但是开启后仍然可以提升攻击难度，总之没有特别的理由，不要关闭它。

**4、X-Content-Type-Options**

互联网上的资源有各种类型，通常浏览器会根据响应头的Content-Type字段来分辨它们的类型。例如：” text/html”代表html文档、” image/png”是PNG图片、”text/css”是CSS样式文档。然而有些资源的content-type是错的或者未定义。这时，某些浏览器会启用MIME-sniffing来猜测该资源的类型，解析内容并执行。

利用浏览器的这个特性，攻击者甚至可能让原本应该解析为图片的请求被解析为JavaScript。

利用下面这个响应头可以禁用浏览器的类型猜测行为：

X-content-Type-Options : nosniff

**5、X-Content-Security-Policy**

这个响应头主要是用来定义页面可以加载哪些资源，减少XSS的发生，详情见CSP

## **CSP (content Security Poliy)**

* 这个规范与内容安全有关，主要用来定义页面可以加载哪些资源，减少XSS的发生。
* Chrome扩展已经引入了CSP，通过mainfest.json中的content\_security\_policy字段来定义。
* 一些现代浏览器也支持通过响应头来定义CSP。
* Firefox和IE支持X-Content-Security-Policy，chrome25和Firefox23开始支持标准的Content-Security-Policy
* 要使用CSP，只需要服务器输出类似下面这样的响应头就行了:

Content-Security-Policy: default-src ‘self’

* CSP指令，多个指令之间用英文分号分割

|  |  |
| --- | --- |
| default-src | 定义针对所有类型（js、image、css、web font、ajax请求、iframe、多媒体等）资源的默认加载策略，某类型资源如果没有单独定义策略，就使用默认的。 |
| script-src | 定义针对JavaScript的加载策略 |
| style-src | 定义针对样式的加载策略 |
| Img-src | 定义针对图片的加载策略 |
| … |  |

* 指令值，多个指令值用英文空格分隔

|  |  |
| --- | --- |
| none | 不允许任何内容 |
| self | 允许来自相同来源的内容 |
| data: | 允许data:协议（如base64编码的图片） |
| www.a.com | 允许加载指定域名的资源 |
| .a.com | 允许加载a.com任何子域的资源 |
| <https://img.com> | 允许加载img.com的https资源（协议需匹配） |
| https: | 允许加载https资源 |
| ‘unsafe-inline’ | 允许加载inline资源（例如常见的style属性，onclick、inline js和inline css等等） |
| ‘unsafe-eval’ | 允许加载动态js代码，例如eval() |

## **浏览器缓存策略（强缓存、协商缓存）**

<https://www.cnblogs.com/bill-shooting/archive/2018/07/21/9347441.html>

**Web缓存的好处**

* 减少网络延迟，加快页面打开速度
* 减少网络带宽消耗
* 降低服务器压力
* ……

**Etag与IF-None-Match**

* Etag与IF-None-Match是一对报文头，属于HTTP 1.1
* Etag是一个响应首部字段，它是根据实体内容生成的一段hash字符串，标识资源的状态，由服务器产生。
* IF-None-Match是一个条件式的请求首部字段。如果请求资源时在请求首部加上这个字段，值为之前服务器返回的Etag。

**Etag能解决什么问题？**

1. Last-modified标注的最后修改时间只能精确到秒级，如果某些文件在1秒钟以内，被修改多次的话，它将不能准确标注文件的新鲜度；
2. 某些文件也许会周期性的更改，但是它的内容不改变（仅仅改变的是修改时间），但last-modified却改变了，导致文件没法使用缓存。
3. 有可能存在服务器没有准确获取文件修改时间，或者与代理服务器时间不一致等情形。

**缓存位置**

浏览器可以再内存、硬盘中开辟一个空间用于保存请求资源副本。

请求一个资源时，会按照优先级（service worker -> memory cache -> disk cache -> push cache）依次查找缓存。

**200 from memory cache**

表示不访问服务器，直接从内存中读取缓存。因为缓存的资源保存在内存中，所以读取速度快，但是关闭进程后，缓存资源也会随之销毁，一般来说，系统不会给内存分配比较大的容量，因此内存缓存一般用于存储较小的文件。同是内存缓存再有效性要求的场景下也很有用（比如浏览器的隐私模式）。

**200 from disk cache**

**C:\Users\shuanggui.yu\AppData\Local\Google\Chrome\User Data\Default\Cache**

表示不访问服务器，直接从硬盘中读取缓存。与内存相比，硬盘的读取速度相对较慢，但硬盘缓存持续的时间更长，关闭进程之后，缓存的资源依然存在。由于硬盘的容量较大，因此一般用于存储大文件。

**200 from prefetch cache**

在preload或prefetch的资源加载时，两者也是均存储在http cache，当资源加载完成后，如果资源是可以缓存的，那么其被存储在http cache中等待后续使用，如果资源不可被缓存，那么其在被使用前均存储在memory cache。

**CDN Cache**

* **浏览器缓存**

当浏览器访问过后的资源，会被浏览器缓存在本地，当下次再访问页面的时候，如果没有过期，就直接读取资源，加快浏览器的加载速度

http缓存机制：

1. Expires：设置最大缓存时间，当时间超过了就去服务器下载
2. http1.1 ，cache-control： max-age =time ， 当time过期后，检测etag 带上etag往服务器发请求，如果etag没变，直接告诉浏览器读本地缓存， 如果没有etag就会检测last-modified，判断 如果上一次更改的时候，距离本次访问时间比较久，说明文件没有发生变化，返回304

* **协商缓存和强制缓存**

协商缓存的标识有2种：ETag/if-None-Match和Last-Modified/if-Modify-Since

浏览器第一次请求一个资源的时候，服务器会告知浏览器一个最后修改时间，当浏览器再次请求的时候，request的请求头会加上if-Modify-Since，服务器收到浏览器发送来的最后修改时间和当前自己的最后修改时间进行对比，一致的话就会返回304，否则，当前自己的最后修改时间大于浏览器携带的最后时间，将返回200，并给浏览器传递新的最后修改时间。

Etag是唯一标识，

文中"只要资源有变化，ETag会重新生成的"。也不能说作者大大说错。而是不严谨。Etag生成规则是由服务器的生成算法决定的，像nginx服务器的Etag生成规则就是由response的 Last-Modified 与 Content-Length 表示为十六进制组合而成，而last-modified又是由mtime生成。当编辑文件但是没修改内容时资源是没有变化的，但是此时mtime已经改变所以Etag也会跟着改变，但是资源文件并没有改变，Etag也会重新生成

协商缓存有两种，一种是Last-Modified，就是第一次请求资源的时候，服务器会在响应头里面设置该字段，表示该资源的最后修改时间，浏览器再第二次请求该资源的时候，会在请求头里面加上一个字段If-Modified-Since，值为第一次请求的时候服务器返回的Last-Modified的值，服务器会判断当前资源的最后修改时间和请求头里面的If-Modified-Since字段是否相同，如果相同，则告诉客户端使用缓存，否则重新下载资源。

另外一种协商缓存是使用Etag，原理与Last-Modified类似，就是第一次请求的时候，服务器会根据资源的内容或者最好修改时间生成一个标识，然后再响应头里面设置Etag返回给客户端，客户端第二次请求的时候会在请求头里面带上这个Etag，也就是再请求头里面加上If-None-Match字段，服务器接收到Etag之后会判断是否与原来第一次的标识相同，如果相同，则告诉客户端使用缓存。

强制缓存：

浏览器在第一次请求资源的时候，服务器响应头里可以设置expires字段，该字段表示该资源的缓存过期时间，第二次请求的时候，如果时间还在该缓存时间之内，则会直接使用缓存，否则重新加载资源。这个expires字段有一个缺陷，就是它必须服务器和客户端的时间严格同步才能生效，所以现在很多人不会使用该方案，另外一种方案是第一次请求资源的时候，服务器设置响应头cache-control：max-age，这样设置的意思是告诉浏览器，这个资源什么时候过期，等第二次请求资源的时候，判断是否超过过期时间，如果没有超出，直接使用缓存。

Cache-control是服务器设置的。

cache-control的其他值： public 客户端和服务端都可以缓存

private 只能客服端缓存

no-store 不使用缓存

no-cache 使用协商缓存