**Performance**

**浏览器的内核**

IE---------->Trident

Safari------>WebKit

Chrome------>Blink

Opera------- >Blink

Firefox------>Gecko

**进程和线程**

1.一个进程有一个或多个线程,

2.现在浏览器:多进程、多线程模型

**浏览器渲染引擎**

一个渲染引擎主要包括：HTML解析器，CSS解析器，javascript引擎，布局(layout)模块，绘图模块

1.阻塞渲染:

1.style标签中的样式：
(1). 由html解析器进行解析；
(2). 不阻塞浏览器渲染（所以可能会产生“闪屏现象”）；
(3). 不阻塞DOM解析；
2.link引入的外部css样式（推荐使用的方式）：
(1). 由CSS解析器进行解析。
(2). 阻塞浏览器渲染(可以利用这种阻塞避免“闪屏现象”)。
(3). 阻塞其后面的js语句的执行
(4). 不阻塞DOM的解析(绝大多数浏览器的工作方式)

3.js阻塞:

因为js可以修改DOM树结构,css样式,以及存在js文件的依赖关系,

所用js阻塞后续DOM解析,页面渲染,和后续js的执行.

2.优化核心理念：尽可能快的提高外部css加载速度

*(1).使用CDN节点进行外部资源加速。*

*(2).对css进行压缩(利用打包工具，比如webpack,gulp等)。*

*(3).减少http请求数，将多个css文件合并。*

*(4).优化样式表的代码*

3.其它知识点:

1.css的解析和js的执行是互斥的（互相排斥），css解析的时候js停止执行，js执行的时候css停止解析。

2.无论css阻塞，还是js阻塞，都不会阻塞浏览器加载外部资源（图片、视频、样式、脚本等）

4.加载完成事件:

1.onload事件:DOM树构建完并且网页所依赖的资源都加载完之后触 发

2.DOMContentLoaded事件:DOM树构建（解析）完成之后

document.addEventListener('DOMContentLoaded',function(){
console.log('3 seconds passed');
});

3.defer属性是在告诉浏览器要等整个页面载入之后，解析完毕才执行该<script…/>元素的脚本

**图层和重绘重排**

1.创建图层的方式:chrome浏览器下

*1. 拥有具有3D变换的CSS属性*

*2. 使用加速视频解码的<video>节点*

*3. <canvas>节点*

*4. CSS3动画的节点*

*5. 拥有CSS加速属性的元素(will-change)*

*推荐使用的方法:1.will-change:transform ;*

*2. transform:translateX(0);*

2.重绘(Repaint):

*重绘是一个元素外观的改变所触发的浏览器行为*

*需要注意的是：重绘是以图层为单位，如果图层中某个元素需要重绘，那么整个图层都需要重绘。*

3.重排(Reflow 又称：回流):

*渲染对象在创建完成并添加到渲染树时，并不包含位置和大小信息。计算这些值的过程称为布局或重排*

*"重排"大多数情况下会导致"重绘"*

4.优化方案:

*1.元素位置移动变换时尽量使用CSS3的transform来代替对top left 等的操作*

*因为transform会避免重绘环节，部分浏览器甚至会避免重排环节。*

*2.【使用opacity来代替visibility】*

*(1).使用visibility不触发重排，但是依然重绘。*

*(2).直接使用opacity即触发重绘，又触发重排*

*(3).opacity配合图层使用，即不触发重绘也不触发重排（大部分浏览器）。即可以加一个will-change:transform ;*

*(4).display:none;即触发重绘，又触发重排*

*3.【不要把获取某些DOM节点的属性值放在一个循环里当成循环的变量】*

*1. offsetTop, offsetLeft, offsetWidth, offsetHeight*

*2. scrollTop/Left/Width/Height*

*3. clientTop/Left/Width/Height*

*4. width,height*

*当你请求上面的一些属性的时候，浏览器为了给你最精确的值，需要刷新内部队列，*

*因为队列中可能会有影响到这些值的操作。即使你获取元素的布局和样式信息跟最近发生或改变的布局信息无关，*

*浏览器都会强行刷新渲染队列。也就是进行了大量的重排*

5.requestAnimationFrame----请求动画帧

说明：该方法会告诉浏览器在下一次重绘重排之前调用这个函数,它与定时器比最大的优势在于不会有抖动,

**let *test*** = ***document***.getElementById(**'test'**)

**let *index*** = 0

**let *id*** = *requestAnimationFrame*(*demo*) //这个方法的参数必须是对象

**function** *demo*() {

***index***++

***test***.**style**.**transform** = **`translateX(**${***index***}**px)`**

***id*** = *requestAnimationFrame*(*demo*) //通过递归实现动画效果

}

*setTimeout*(()=>{

*cancelAnimationFrame*(***id*** ) //停止请求动画帧事件

},2000)

**CDN**

CDN是一组分布在多个不同地理位置的web服务器，用于更加有效的向用户发布内容

**函数防抖,节流**

1.防抖: 动作结束时只执行一次,清除中间的次数

概念： 延迟要执行的动作，若在延时的这段时间内，再次触发了，则取消之前开启的动作，重新计时。即在动作结束后只执行一次

使用定时器实现:触发事件时,清除定时器,再执行定时器.

2.节流:有一个下限,超出下限就不会平凡触发.

概念：设定一个特定的时间，让函数在特定的时间内只执行一次，不会频繁执行

实现：定时器、标识 ;

**let *isCanLog*** = **true**

***document***.**onscroll** = ()=>{

**if**(***isCanLog***){

***console***.log(1)

***isCanLog*** = **false**

*setTimeout*(()=>{

***isCanLog*** = **true**

},2000)

}

}

**浏览器的本地存储**

1.浏览器本地存储一共有3种方式:

Cookie , SessionStorage , LocalStorage

注意前端讲的session指SessionStorage ;而后端的session指session会话, 两个东西不一样.

Cookie的流程: (存储空间小4kb)

1.客户端发送请求给服务端,

2.服务端接收用户数据,存储在session会话,再把id通过cookie返回给客户端,

3.客户端下次访问会带着cookie访问服务端,服务器就可以在数据库找到用户信息.

SessionStorage:存储的内容会随着浏览器窗口关闭而消失。

LocalStorage:存储的内容，需要手动清除才会消失。

SessionStorage,LocalStorage存储内容大小一般支持5-10MB

1.如何查看SessionStorage和LocalStorage:

chrome浏览器控制台:Application

2.如何使用:增删改查

LocalStorage和SessionStorage方法一样

1.增: localStorage.setItem('key', 'value'); //注意存储时如果是对象,需要用JSON.stringify(obj)转化为字符串.

2.查:var data = xxxxxStorage.getItem('person'); //读取时需要用JSON.parse(str)转化为对象.

3.删:xxxxxStorage.removeItem('key');

4.清空: xxxxxStorage.clear()

2.跨页签通讯:(数据同步)

1.storage事件

*window.addEventListener('storage',function (event) {}*

*window.onstorage = function(event){}*

2.event中存储很多数据,重点关注:

*key : 修改或删除的key值，如果调用clear(),为null*

*newValue: 新设置的值，如果调用clear(),为null*

*oldValue:调用改变前的value值,如果调用clear(),为null*

*url : 触发该脚本变化的文档的url*

*storageArea : 当前的storage对象*

注意跨页签通讯必须是同一个ip

**缓存:节省宽带资源,减轻服务器压力**

1.缓存分类

强缓存:不会向服务器发送请求，直接从本地缓存中获取数据

状态码为: 200

expires:(http1.0时的规范)它的值为一个时间,在这时间前访问的,一直走缓存,在这个时间后才会走服务器.

cache-control:(http1.1时的规范)主要是利用该字段的max-age = number 值来进行判断, 通过资源第一次的请求时间和Cache-Control设定的有效期，计算出一个资源过期时间 , 再拿这个过期时间跟当前的请求时间比较，如果请求时间在过期时间之前，就能命中缓存，否则就不行

cache-control比expires权重高,所已通过cache-contro = max-age = 0;可以清除强缓存

协商缓存:向服务器发送请求，服务器会根据请求头的资源判断是否命中协商缓存,命中走缓存 状态码为:304(通知浏览器从缓存中读取资源)

Last-Modified/If-Modified-Since:两个都是时间,

判断过程

1.浏览器第一次跟服务器请求一个资源，服务器在返回这个资源的同时加上Last-Modified,表示这个资源在服务器上的最后修改时间

2.浏览器再次跟服务器请求这个资源时会加上If-Modified-Since,表示上一次请求时返回的Last-Modified的值

3.服务器再次收到资源请求时,比较Last-Modified和If-Modified-Since,如果没有变化则返回304走缓存, 否则服务器返回数据并更新Last-Modified的值.

Etag/If-None-Match:返回id

这两个值是由服务器生成的每个资源的唯一标识字符串，只要资源有变化就这个值就会改变

其判断过程与Last-Modified/If-Modified-Since类似

好处:如果只是时间判断,频率过快就会出现问题,但是id不会出问题利用Etag能够更加准确的控制缓存

**问题:强缓存如何重新加载新的资源**

通过更新页面中引用的资源路径，让浏览器主动放弃加载缓存去加载新的资源

实例:解决ie Ajax请求的强缓存

**"http://localhost:3000/get\_verify\_code?" + *Date***.now()