前端性能优化
及js开发常见优化总结

Marson（5.01号）

一、从输入URL发生了什么以及如何优化

1.URL 解析

2.DNS 解析

3.TCP 连接

4.发送请求

5.接受响应

6.渲染页面

1.URL解析

1.判断输入的是关键字搜索还是url访问
2.解析url: 划分协议域名(top个人、com国际、cn中国 、gov政府、org官方 、net系统 、io博客),
端口号(http:80,https:443,ftp:21) ,
请求路径
请求参数编码

3.启用 HSTS (HTTP Strict Transport Security)

2.DNS解析

- 简述:
- 主要是解析你的域名 通过Ip定位到到你存放资源服务器

- DNS查询方式:
- 本地浏览器缓存（chrome://net-internals/#dns地址查看）->系统缓存（可以看看本地host文件）->路由器缓存->ISP 缓存->走服务器的查询（1.迭代查询、2.递归查询）

2.性能优化从何入手-DNS预解析

1. <meta http-equiv="x-dns-prefetch-control" content="off">

content="off"关闭隐式预解析

隐式预解析（默认情况下，对于a标签来说，浏览器会对当前页面中与当前域名不在同一个域的域名进行预获取，并且缓存结果）但是对于https就失效了使用方式

<meta http-equiv="x-dns-prefetch-control" content="on">

2. <link rel="dns-prefetch" href="http://www.zhaowaedu.com" />

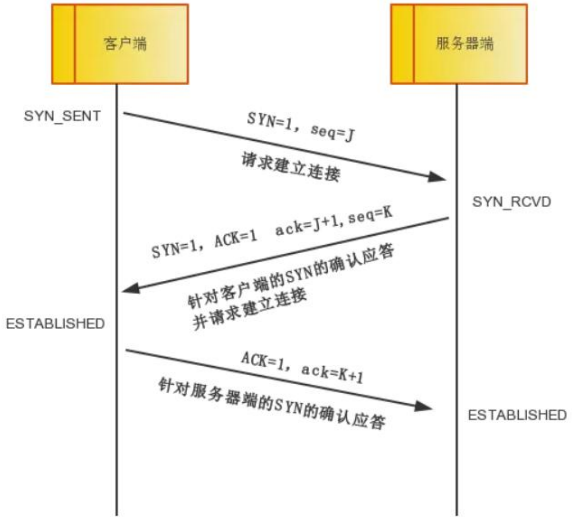
不知道协议情况下：

<link rel="dns-prefetch" href="//renpengpeng.com" />

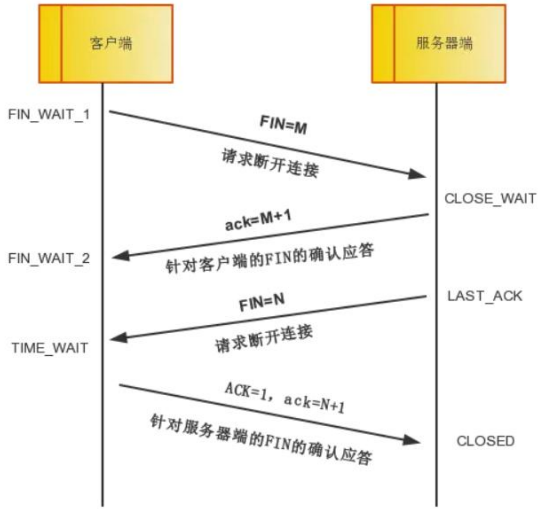
不要写重复的预解析

3.TCP连接

- 三次握手



四次挥手



相关经典面试题

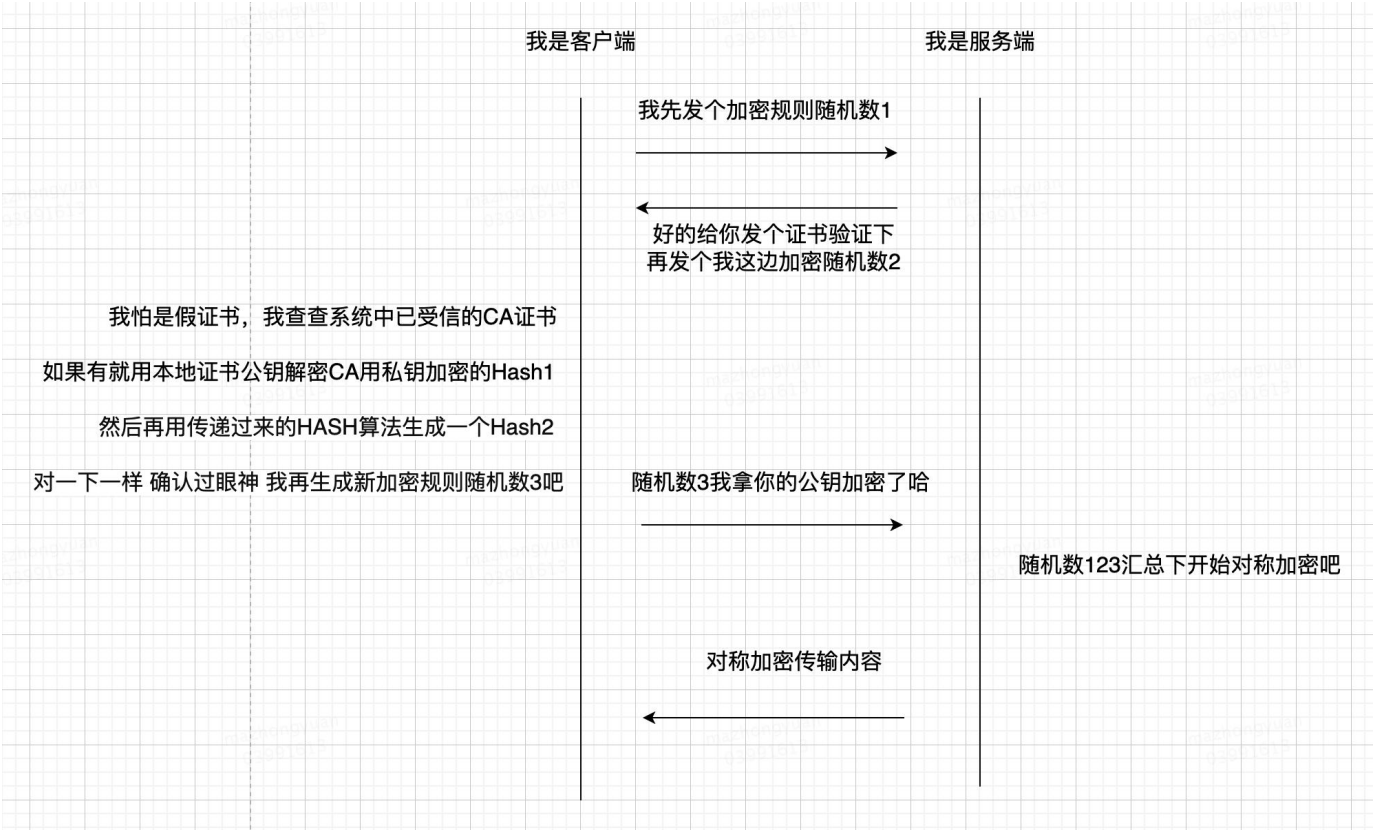
1.什么是三次握手，四次挥手？
2.为什么需要四次挥手呢？
3.什么是https，以及ssl加密是什么？

https流程总结

1.对称加密

2.非对称加密

3.ssl加密证书（1.颁发机构CA，2.有效期，3.服务端公钥，4.服务端Hash算法，5.签名）


我是客户端
我是服务端
我先发个加密规则随机数1
好的给你发个证书验证下
再发个我这边加密随机数2
我怕是假证书，我查查系统中已受信的CA证书
如果有就用本地证书公钥解密CA用私钥加密的Hash1
然后再用传递过来的HASH算法生成一个Hash2
对一下一样 确认过眼神 我再生成新加密规则随机数3吧
随机数3我拿你的公钥加密了哈
随机数123汇总下开始对称加密吧
对称加密传输内容

4&5发送请求接收响应

将cookie等信息放到请求头中 （减少cookie信息）
各个浏览器请求的并发数量如下 （减少http请求，资源合并，懒加载，CSS Sprite）

浏览器	HTTP 1.1
IE 6, 7	2
IE 8, 9	6
Firefox 13	6
Chrome 20	6
Safari 5.1.7	6
Opera 11.64	8

4&5发送请求接收响应

利用好缓存：强缓存，协商缓存，本地存储，serviceworker等

常问面试题：

1.强缓存，协商缓存是什么？

2.localstorage和sessionstorage的区别？

6 渲染页面及优化方案

script：js加载阻塞dom和cssmos执行，因此js放下边，css放在上边

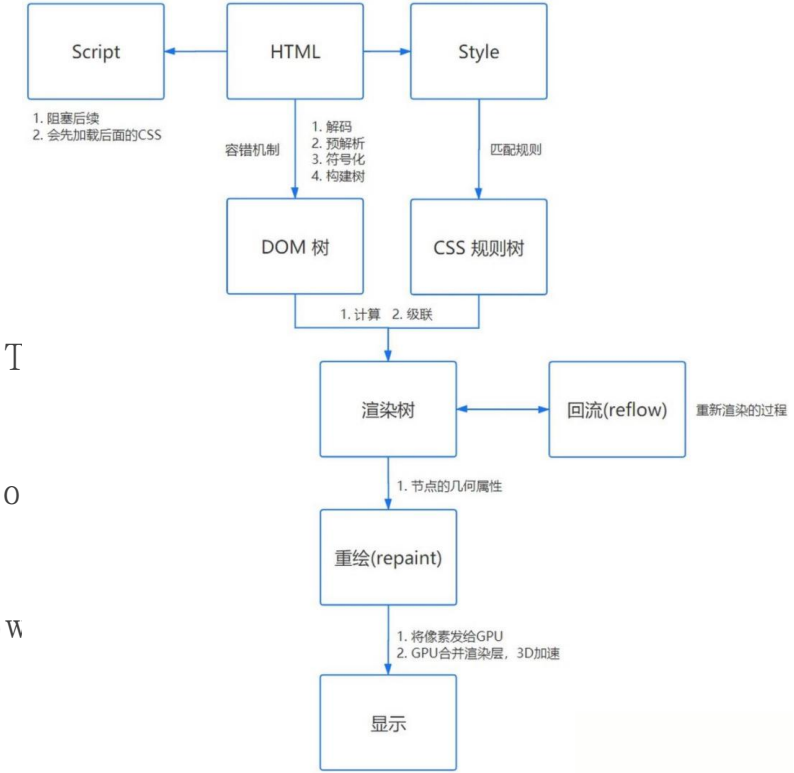
预解析：

preconnect：<link rel="preconnect" href="//zhaowa.com">（将会做 DNS 解析，T TCP 握手）

preload：<link rel="preload" href="/path/to as="style">

prefetch：<link rel="prefetch" href="/zhaow as="script">

prerender：<link rel="prerender" href="//zhaowa.com/zhaowa.html">

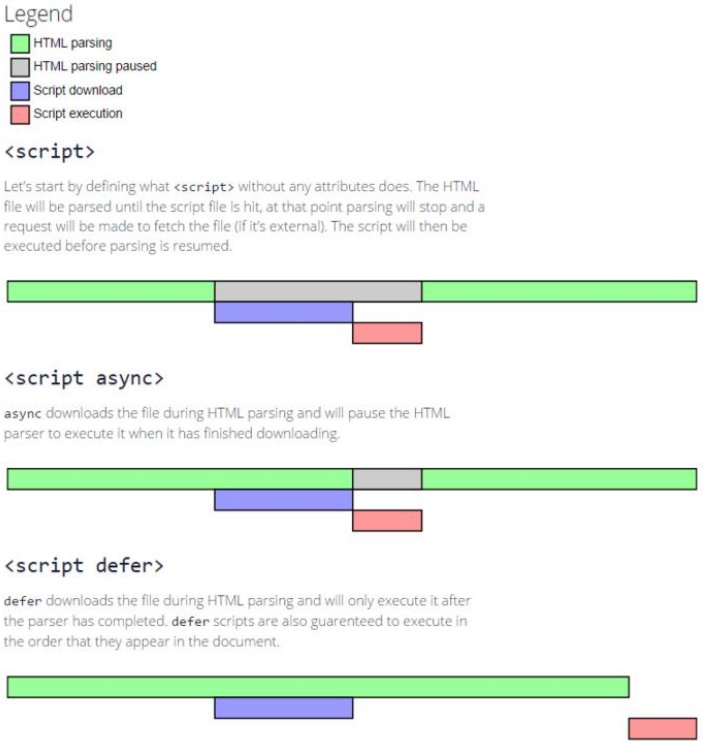


6 渲染页面及优化方案

异步加载js使用async defer优化js加载

```
<script async
src="script.js"></script>
```

```
<script defer
src="script.js"></script>
```



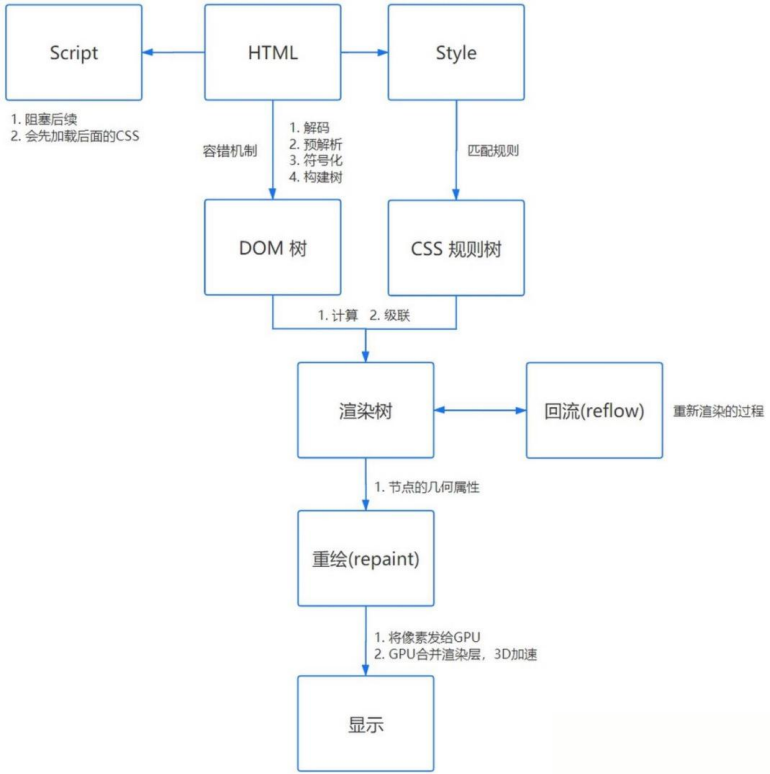
6 渲染页面及优化方案

css：异步加载，根据css规则解析出所有的 CSS 并进行标记化 尽量写class id 避免过度多层查找，不太推荐图片/字体等转成base64放到css

DOM去匹配css rule的时候必须先等页面的css都下载完成 因此css放到header中

尽量减少dom操作，少写一些改变布局的样式 防止回流

多写css3 的属性如transform，translate，opacity，减少重排重绘



文档

雅虎军规
中文翻译：https://github.com/creeperyang/blog/issues/1

二、js开发中常见优化总结

1.尽可能的用局部变量来代替全局变量（解析器直接查找作用域中的对象）

```
function test(){
    var name = “lisa”
}
```

2.我们写for循环时候，缓存多次使用的对象或者属性

```
var num = obj.name
for( var i=0;i<num;i++){
}
```

二、js开发中常见优化总结

3.访问属性a.b 速度快于a.b.c 因此尽量嵌套不要太深
4.短路表达式
5.写递归一定要定义好边界条件
6.注意隐式转换

```
console.log(1+” zhaowa” )
console.log(1+true)
console.log( “zhaowa” +true)
console.log(1+” undefind” )
console.log(1+” null” )
```

二、js开发中常见优化总结

7.尽量避免使用闭包

8.声明多个变量时使用单个语句

```
let a=1,b=2,c=3;
```

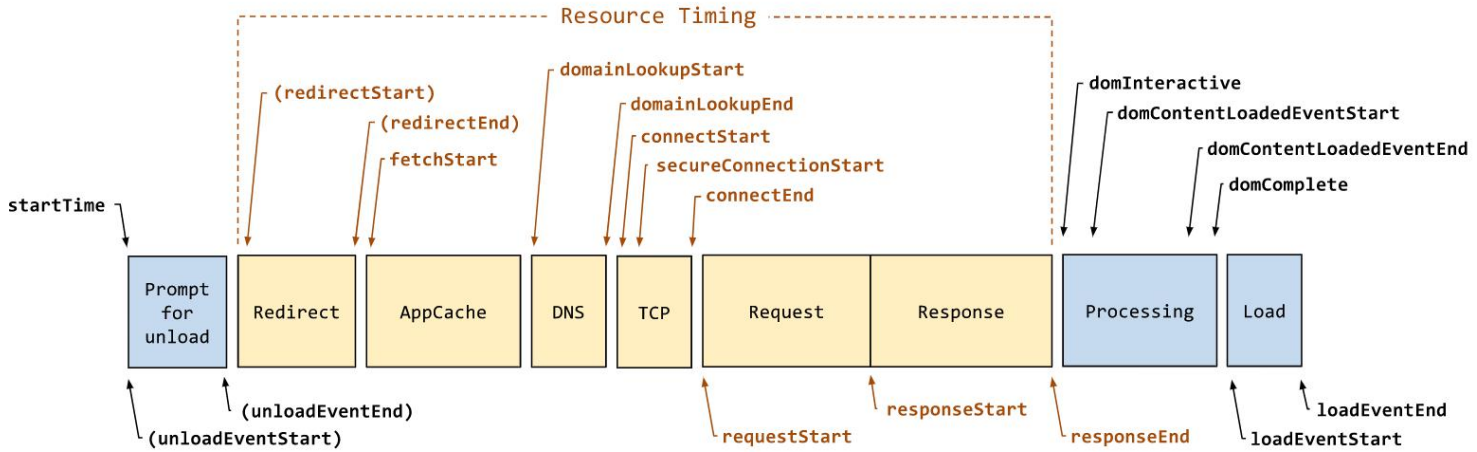
9.面对大量if else 可以定义对象的方式去写

```
var obj={“red”:red,”black”:black,”yellow”:yellow}
if (color in colorObj) {
        obj[color]();
    }
```

10.三元表达式以及es6里的语法

三、大厂性能的计算方式以及优化方案

Navigation Timing Level 2



1.各个时间点解释

unloadEventStart-unloadEventEnd:上一个页面卸载开始到卸载结束
redirectStart-redirectEnd: 重定向开始到结束
fetchStart: 页面加载
domainLookupStart - domainLookupEnd: DNS查找开始到结束
connectStart - connectEnd: tcp链接开始到结束
secureConnectionStart: https开始链接时间
requestStart : 发起请求时间
responseStart: 首字节时间
responseEnd: 请求结束时间
domInteractive: dom解析结束
domContentLoadedEventStart 在DOM树解析完成后，网页内资源加载开始的时间
domContentLoadedEventEnd DOM树解析完成后，网页内资源加载完成时间
domCompelete Dom树解析完成，且资源也准备就绪的时间，Document.readyState变成complete.并将抛出readystatechange 事件
loadEventStart load 事件发送给文档，也即load回调函数开始执行的时间
loadEventEnd load回调函数执行完成的时间

1.各个时间点解释

```
let t = performance.timing
console.log('DNS查询耗时 ： ' + (t.domainLookupEnd - t.domainLookupStart))
console.log('TCP链接耗时 ： ' + (t.connectEnd - t.connectStart))
console.log('request请求耗时 ： ' + (t.responseEnd - t.responseStart))
console.log('解析dom树耗时 ： ' + (t.domComplete - t.domInteractive))
console.log('白屏时间 ： ' + (t.responseStart - t.navigationStart).toFixed(0))
console.log('domready时间 ： ' + (t.domContentLoadedEventEnd -
t.navigationStart))
console.log('onload时间 ： ' + (t.loadEventEnd - t.navigationStart))
```

2.关键时间点

1. 白屏时间 - responseStart节点

2. DOM Ready-domContentLoadedEventEnd节点

3. DOM完全加载完时间点- domComplete节点

4. 首屏时间：用户看到第一屏页面的时间 （比较难算）

5. onload：原始文档和所有引用的内容已经加载完成。用户最明显的感觉就是浏览器上Loading状态结束 loadEventEnd 节点

2.关键时间点

FP (First Paint) 首次绘制

FCP (First Contentful Paint) 首次内容绘制

LCP (Largest Contentful Paint)` 最大内容渲染

DCL (DomContentloaded)

L (onLoad)

3.大厂首屏计算方式