前端性能优化

为什么需要前端性能优化?

遇到一个页面，5秒还没加载完成，那个转啊转，或者页面完全白屏，那简直把人逼疯了。从用户体验的角度看，前端性能优化是非常有必要的。网页最长加载时间一般不能超过3秒。

性能优化可以从哪几个方面入手?

首先我们需要确定网页的性能指标，可量化的目标以及可持续跟踪的优化数据是性能优化工作得以持续进行的保障，同时也是源动力！比如：

1. 首屏加载时长
2. DOM加载时长
3. 页面白屏时长

我们一般通过三种方式来检验我们的网页性能：

1. 通过浏览器开发者工具或浏览器插件、Fiddler、Charles等查看页面加载情况。原理是通过追踪HTTP请求与响应的时间，以图形的方式列出所有资源的下载情况。缺点是人为操作，难以实现批量测试与统计。

2. 在页面中引入额外的代码钩子来记录时间等相关数据。缺点是加重了开发者与测试人员的负担，还有可能因为检测代码本身的潜在问题影响页面的性能。如果好一点的话，会接入一个性能数据收集系统，采取并分析数据。

3. 使用第三方的工具如Page Speed、YSlow和WebPagetest，能够选择在不同浏览器和不同地域进行测试，并且给出各方面的评分以及提供一些优化建议。但某些服务需要排队等待，并且难以实现批量测试与统计。下面是使用WebPagetest测试京东首页的情况：

人人都知道雅虎军规！

网页内容	服务器	Cookie	图片
减少http请求次数	使用CDN	减少Cookie大小	优化图像
减少DNS查询次数	添加Expires或Cache-Control报文头	页面内容使用无cookie域名	优化CSS Sprite
避免页面跳转	Gzip压缩传输文件	**CSS**	不要在HTML中缩放图片
缓存Ajax	配置ETags	将样式表置顶	使用小且可缓存的favicon.ico
延迟加载	尽早flush输出	避免CSS表达式	**移动客户端**
提前加载	使用GET Ajax请求	用<link>代替@import	保持单个内容小于25KB
减少DOM元素数量	避免空的图片src	避免使用Filters	打包组建成符合文档
根据域名划分内容		**Javascript**	
减少iframe数量		将脚本置底	
避免404		使用外部Javascript和CSS文件	
		精简Javascript和CSS	
		去除重复脚本	
		减少DOM访问	
		使用智能事件处理	

以下，我们从服务端、网络、客户端三个方面来一一突破速度性能的提升。

服务端

使用内容分发网络（Content Delivery Network，CDN）

通过在现有的Internet中增加一层新的网络架构，将网站的内容发布到最接近用户的 cache服务器内，通过DNS负载均衡的技术，判断用户来源就近访问cache服务器取得所需的内容。深圳用户访问遥远的美国服务器，当然不理想了。把静态内容分布到CDN可以减少用户响应时间20%或更多。

静态资源缓存，移动端离线缓存

如果可以减少服务端的负担，在应用离线时可使用资源或加载资源更快，岂不乐哉？ 缓存利用可包括：添加 Expires 头，配置 ETag，使 Ajax 可缓存等。其实，恰当的缓存设置可以大大的减少 HTTP请求，也可以节省带宽 。

- 配置 ETag：即If-None-Match: 上次 ETag 的内容。浏览器会发出请求询问服

务端，资源是否过期；服务端发现,没有过期，直接返回一个状态码为 304、正文为空的响应，告知浏览器使用本地缓存；如果资源有更新，服务端返回状态码 200、Etag 和正文。这个过程被称之为 HTTP 的协商缓存，通常也叫做弱缓存。

- 添加 Expires 头：服务端通过响应头告诉浏览器，在什么时间之前（Expires）或在多长时间之内（Cache-Control: Max-age=xxx），不要再请求服务器了。这个机制我们通常称之为 HTTP 的强缓存。一般会对 CSS、JS、图片等资源使用强缓存，而入口文件（HTML）一般使用协商缓存或不缓存。
- AppCache：AppCache主要利用manifest 文本文件，告知浏览器被缓存的内容以及不缓存的内容。
- manifest 文件可分为三个部分：（1） CACHE MANIFEST – 在此标题下列出的文件将在首次下载后进行缓存，等价于CACHE（2） NETWORK – 在此标题下列出的文件需要与服务器的连接，且不会被缓存（3） FALLBACK – 在此标题下列出的文件规定当页面无法访问时的回退页面使用AppCache方案的步骤：（1） 整理出需要缓存的静态文件列表，如juqery.js和gb.css。（2） 配置服务器支持。（3） 确定内容更新机制和浏览器兼容方案。
- LocalStorage：用于持久化的本地存储，除非主动删除数据，否则数据是永远不会过期的。 ## 网络 ### 减少请求数 可通过以下方式减少请求数：
- 小图片合并雪碧图；
- JS、CSS文件选择性合并；
- 避免重复的资源请求。

减少请求数对于速度优化来说最重要最有效的，特别是网络差的用户。一个完整的请求需要经过域名解析以及DNS寻址、与服务器建立连接、发送数据、等待服务器响应、接收数据的过程；每个请求都需要携带数据，因此每个请求都需要占用带宽；浏览器进行并发请求的请求数是有上限的。请求多了的情况，明显增加了网页的响应时间。一个页面由多个模块拼接而成，几个模块中请求了同样的资源时，就会导致资源的重复请求。

减少文件大小（减少请求带宽）

1. 压缩CSS、JS、图片；
2. 尽可能控制DOM节点数；
3. 精简css、 JavaScript，移除注释、空格、重复css和脚本。
4. 开启Gzip，Gzip的思想就是把文件先在服务器端进行压缩，且压缩率达到85%，然后再传输，传输完毕后浏览器会 重新对压缩过的内容进行解压缩，并执行。。好处在于Gzip的支持已经很好，且爬虫可识别，压缩率达到

66%-85%显著减少了文件传输的大小。另外，gzip对pdf文件的压缩效果不大，而且会浪费CPU。

合理使用静态资源域名

> 域名的要求是短小且独立。

短小可以减少头部开销，因为域名越短请求头起始行的 URI 就越短。之所以要求独立，因为独立域名不会共享主域的 Cookie，可以有效减小请求头大小，这个策略一般称之为 Cookie-Free Domain；另外一个原因是浏览器对相同域名的并发连接数限制，一般允许同域名并发 6~8 个连接，域名不是越多越好，每个域名的第一个连接都要经历 DNS 查询（DNS Lookup），导致会耗费一定的时间，控制域名使用在2-4个之间。

另外注意：同一静态资源在不同页面被散列到不同子域下，会导致无法利用 HTTP 缓存。

使用HTTP 2

HTTP 2 相比 HTTP 1.1 的更新大部分集中于：

- 多路复用：多路复用很好地解决如何让重要资源尽快加载这个问题。同域名下或者不同域但是同时满足同一个 IP以及使用同一个证书的这两个条件中的所有通信都在单个连接上完成，此连接上同时打开任意数量的双向数据流（HTTP 1.1 有连接数限制）。使用多域名加上相同的 IP 和证书部署 Web 服务有特殊的意义：让支持 HTTP/2 的终端只建立一个连接，用上 HTTP/2 协议带来的各种好处；而只支持 HTTP/1.1 的终端则会建立多个连接，达到同时更多并发请求的目的。
- HEAD 压缩：HTTP/2 将请求和响应数据分割为更小的帧，并对它们采用二进制编码（ Binary Framing ）。在 HTTP/1 中，HTTP 请求和响应都是由「状态行、请求 / 响应头部、消息主体」三部分组成，状态行和头部却没有经过任何压缩，直接以纯文本传输。如下图的比较：
- 在 HTTP/2 中，每个数据流都以消息的形式发送，而消息又由一个或多个帧组成。多个帧之间可以乱序发送，因为根据帧首部的流标识可以重新组装。
- 请求优先级：服务器可以根据流的优先级，控制资源分配(CPU、内存、带宽)，而在响应数据准备好之后，优先将最高优先级的帧发送给客户端。
- 服务器推送：启动Server Push，意味着服务端可以在发送页面HTML时主动推送其它资源，有自己独立的URL，可以被浏览器缓存；如果服务端推送的资源已经被浏览器缓存过，浏览器可以通过发送 RST_STREAM 帧来拒收。

客户端

1. 使用外链CSS和JS，CSS放头，JS放尾，防止阻塞以减少对并发下载的影响，尽早刷新文档的输出。
2. html的代码优化
3. 避免空的图片src
4. 协议自适应，减少html文件大小，将https://和http://都替换成//
5. css的代码优化
    - 建议使用类选择器，访问比较快；
    - 不建议使用很长的base64；
    - 避免CSS表达式；
    - 避免使用Filters。
6. js的代码优化
    - 避免使用eval和width；
    - 减少作用域链查找；
    - 减少DOM访问，尽量缓存DOM；
    - 充分利用事件委托；
    - 减少Repaint（重绘）和Reflow（重排）最好通过批量更新元素减少重排次数，如设置类class统一更新样式，在添加多个li
    - 元素将会触发多次页面重排的情况下使用 DOM fargment 在内存中创建完整的 DOM 节点，然后再一次性添加到 DOM 中。
7. 图片格式的选择：
    - 颜色较为丰富的图片而且文件比较大的（40KB – 200KB）或者有内容的图片优先考虑 jpg；图标等颜色比较简单、文件体积不大、起修饰作用的图片，优先考虑使用 PNG8 格式；图像颜色丰富而且图片文件不太大的（40KB 以下）或有半透明效果的优先考虑 PNG24 格式。
    - 条件允许的，使用新格式WEBP和BPG。
    - 用SVG和ICONFONT代替简单的图标。
    - 用字蛛来代替艺术字体切图，它可剔除没有使用的字符，从而解决中文字体过大的问题，并编码成跨平台兼容的格式。
8. 合理分配资源加载时间，按需加载，包括CSS、JS文件以及图片、业务模块等。根据我们网页最初加载需要的最小内容集推断其他内容延迟加载；无条件提前加载公共内容或根据用户行为推断提前加载某些内容，如根据搜索框输入的文字来判断加载的内容。加载机制如下：
    - 预加载
    - Dom Ready后加载
    - onLoad后加载

- 滚动加载
- 减少DNS 查询：DNS 查询一般需要几毫秒到几百毫秒，移动环境下会更慢。我们可以预先读取DNS，减少用户等待时间。

前端安全

常见攻击：

XSS (Cross Site Script)，跨站脚本攻击

往Web页面里插入恶意html代码。特点是攻击者的代码必须能获取用户浏览器端的执行权限，要杜绝此类攻击出现可以在入口和出口进行严格的过滤。三种类型：

1. 反射型XSS：一次性；将包含注入脚本的恶意链接发送给受害者。
2. 持久型XSS：用户输入的数据“存储”在服务器端，比如一条包含XSS代码的留言。

3. DOM XSS：使用一些eval等有输出的语句意味着多了一份被XSS的风险。应对策略：。当恶意代码值被作为某一标签的内容显示：在不需要html输入的地方对html 标签及一些特殊字符(” < > & 等等)做过滤，将其转化为不被浏览器解释执行的字符。。当恶意代码被作为某一标签的属性显示，通过用“将属性截断来开辟新的属性或恶意方法：属性本身存在的 单引号和双引号都需要进行转码；对用户输入的html 标签及标签属性做白名单过滤，也可以对一些存在漏洞的标签和属性进行专门过滤。

4. CSRF(Cross Site Request Forgery)，跨站点伪造请求，通过伪造连接请求在用户不知情的情况下，让用户以自己的身份来完成攻击者需要达到的一些目的。

5. cookie劫持，通过获取页面的权限，在页面中写一个简单的到恶意站点的请求，并获取用户的cookie登录某些站点。

对于crsf 和cookie 劫持的策略：

1. 通过 referer、token 或者 验证码 来检测用户提交。
2. 尽量不要在页面的链接中暴露用户隐私信息。
3. 对于用户修改删除等操作最好都使用post 操作 。
4. 避免全站通用的cookie，严格设置cookie的域。

数据通道安全

国内的众多网站都没有实现全站HTTPS。这是目前为止最重要的一步，所有的数据在发送之前就会被加密，攻击者无法查看或篡改数据包的内容。HTTPS可以理解为HTTP+SSL/TLS，通过数据加密、校验数据完整性和身份认证三种机制来保障安全。HTTPS的缺点是网站在加上TLS证书时，可能导致RTT往返时延增加，并且 HTTPS通信过程的非对称和对称加解密计算会产生更多的服务器性能和时间上的消耗，但是这是可以优化的，这里就不细说了。

浏览器安全

同源策略

首先了解一下同源策略：

- 源指的是有相同的HOST、相同的协议、相同的端口。
- 同源策略以源为单位，把资源天然分隔，保护了用户的信息安全。
- 绕过同源策略让javascript访问其他源的资源的方法，如：JSONP、CORS、flash等。
- 同源策略不是绝对安全的，面对很多攻击是无能为力的，比如XSS，因为此时攻击者就在同源之内。

不建议使用JSONP，因为JSONP通常在脚本中写一个回调函数，然后把回调函数的名字写在请求的URL中，因此如果请求数据的服务器被黑了，那么黑客就能在返回的数据中植入恶意代码，从而窃取用户的隐私信息。

跨域资源共享CORS允许资源提供方在响应头中加入一个特殊的标记，使你能通过XHR来获取、解析并验证数据。这样就能避免恶意代码在你的应用中执行。在响应头中加入的标记如下：

```
Access-Control-Allow-Origin: allowed origins
```

如果对Access–Control-Allow-Origin设置为*其实是比较危险的，如果没有携带会话认证意味着信息被公开在全网，建议设置具体的域名，而且跨域的时候记得带上session id；严格审查请求信息，比如请求参数，还有http头信息，因为 http头可以伪造。

CSP(Content Security Policy)

CSP指定网站上所有脚本和图片等资源的源站点，也能阻止所有内联（inline）的

脚本和样式。即使有人在页面评论或者留言中嵌入了脚本标签，这些脚本代码也不会被执行。可通过两种方式设置，如果 HTTP 头与 Meta 定义同时存在，则优先采用 HTTP 头中的定义：

- 通过 HTTP 头，比如只允许脚本从本源加载：Content-Security-Policy: script-src 'self'，其中script-src 'self'是策略。
- 通过HTML的Meta标签，比如只允许脚本从本源加载 `<meta http-equiv="Content-Security-Policy" content="script-src 'self'">`

其他策略：

1. script-src – 设置可以接受的JavaScript代码的源站点
2. style-src – 设置可以接受的CSS样式代码的源站点
3. connect-src – 定义浏览器可以通过XHR、WebSocket或者 EventSource访问哪些站点
4. font-src – 设置可以接受的字体文件的源站点
5. frame-src – 定义浏览器可以通过iframe访问哪些站点
6. img-src – 设置可以接受的图片的源站点
7. media-src – 设置可以接受的音频和视频文件的源站点
8. object-src – 设置可以接受的Flash和其它插件的源站点

缺点：默认情况下，所有的内联JavaScript脚本都不会被执行，因为浏览器无法区分自己的内联脚本和黑客注入的脚本。CSP还会默认阻止所有eval()风格的代码的执行，包括setInterval/setTimeout中的字符串和类似于new Function('return false')之类的代码。

iframe 沙箱环境

利用iframe进行跨源：HTML5为iframe提供了安全属性 sandbox，iframe的能力将会被限制。

Secure和HttpOnly属性

Secure能确保cookie的内容只能通过SSL连接进行传输。Secure和HttpOnly属性告诉浏览器cookie的内容只能分别通过HTTP(S)协议进行访问，从而避免了被轻易窃取，比如禁止从JavaScript中的document.cookie访问，因此cookie在浏览器document中不可见了。如果单独使用的话，无法全面抵御跨站点脚本攻击，通常和其他技术组合使用。使用方法如下：

```
Set-Cookie: <name>=<value>[; <name>=<value>] [; expires=<date>][; dom
ain=<domain_name>][; path=<some_path>][; secure][; HttpOnly]
```

其他安全相关的HTTP 头

1. X-Content-Type-Options 告诉浏览器相信此服务器下发的资源的类型，防止类型嗅探攻击。
2. HPKP(Public Key Pinning) Public Key Pinning 是一个response 头，用来检测一个证书的公钥是否发生了改变，防止中间人攻击。
3. HSTS (HTTP Strict-Transport-Security) 强制使用TSL作为数据通道。

HTML5 对web安全的影响

html5有很多新的特性能力，然而能力越大，被攻破后的危险就越大。HTML5 对xss的影响主要体现在:

- 攻击面更大，html5带来更多的标签和更多的属性如`<video>,<audio>,<canvas>`等；
- 危害更大，HTML5更多的资源可以被xss利用。黑客可以利用浏览器的一切权限，比如本地存储、GEO、服务器推送机制WebSocket，js多线程执行Webworker等。

比如localstorage只能通过js设置和获取，导致的结果是不能像cookie一样设置httponly等属性，所以localstorage中不能存放敏感信息，最好能够在服务端进行加密，可以配合CORS来获取网站的localstorage的信息。

响应式

响应式布局简而言之，就是一个网站能够兼容多个终端，可以为不同终端的用户提供更加舒适的界面和更好的用户体验。

基于栅格布局规划响应式设计，每个模块尽可能严格遵循栅格布局，符合栅格的小模块能很灵活的适应多个分辨率的展示。

1. 拥抱flexbox。
2. 使用动态的字体大小单位+rem单位使用。
3. 使用CSS3 mediaQuery 技术响应用户设备。
4. 利用百分比。
5. 对低版本浏览器使用JS动态响应。
6. 一套“自适应”素材兼容各种分辨率，提升页面性能，比如自适应的图片/视频

素材。

兼容性

估计很多人对这句话都有体会：IE虐我千百遍，我待IE如初恋。当然，除了 IE 上有兼容性问题，其他浏览器比如 Android 上的低版本浏览器也有较多问题。

是否继续保持对低端浏览器的兼容性，我们可以用数据跟产品经理或者老板说话，减少我们的工作量，最好在项目之前就定下来支持最低支持的版本是什么，然后设计一个对应兼容方案。

兼容性的原则

优雅降级/平稳退化 就是说，在低级浏览器能够保证其可用性和可访问性；Web站点在所有新式浏览器中都能正常工作，如果用户使用的是老式浏览器，则代码会检查以确认它们是否能正常工作。由于IE独特的盒模型布局问题，针对不同版本的IE的hack实践过优雅降级了,为那些无法支持功能的浏览器增加候选方案，使之在旧式浏览器上以某种形式降级体验却不至于完全失效.

渐进增强在保证代码、页面在低级浏览器中的可用性及可访问性的基础上，逐步增加功能及用户体验。从被所有浏览器支持的基本功能开始，逐步地添加那些只有新式浏览器才支持的功能,向页面增加无害于基础浏览器的额外样式和功能的。当浏览器支持时，它们会自动地呈现出来并发挥作用。

如何解决兼容问题

1. 确认触发场景，什么浏览器、版本、什么情况下会出现这个问题，做到稳定复现。
2. 找到问题原因，为什么会出现这样的问题（自己琢磨、网上搜、问同事）。
3. 确定解决办法：参考现成的规范，比如某些属性不能使用以及一些hack的处理。
4. 积累兼容性处理方法。

搜索SEO

语义化

1. 标签语义化对搜索引擎友好，良好的结构和语义容易被搜索引擎抓取。
2. 善用标题h1，h2，h3，h4，h5，h6，特别是h1和h2；H(x)标签中使用关键字，可提升排名。同时设置 rel=“nofollow”避免权重流失。
3. 使用 HTML5 中的 Microdata 对 Web 页面上已经存在的数据提供附加的语义。Microdata 由名字 / 值（name/value）对组成，每一个词汇表定义一组命名的属性。对 Microdata 的支持可以影响搜索结果的显示，使得显示结果更加丰富，虽然不能影响搜索结果的排名，但是网站的流量可能会有所增加。类似的技术还有资源描述框架RDF、微格式Microformat 。

关键词优化

1. 站点内容以及关键词的选择。
2. 描述标签、关键词标签、代替属性。
3. 长尾关键词：非目标关键词但也可以带来搜索流量的关键词；例如，目标关键词是服装，其长尾关键词可以是男士服装、冬装、户外运动装等。长尾关键词基本属性是：可延伸性，针对性强，范围广。
4. 关键词的分布情况。
5. 关键词密度、看重：合理的关键字密度可获得较高的排名位置，密度过大会起到相反的效果。一般说来，在大多数的搜索引擎中，关键词密度在2%~8%是一个较为适当的范围，有利于网站在搜索引擎中排名。
6. 是否存在作弊行为。搜索引擎降权过渡优化网站(降低排名)。

链接

1. 优化文件目录结构和URL。URL应该有语义性，简短易懂。
2. 通过推广暴露自己的链接，增加信任度。链接分为外向链接和内向（反向）链接，外向链接就是从本站点到其他站点，内向链接就是从其他站点到我的站点，可以尝试使用反向链接生成器。或者通过写软文、发布分类信息、发布博客文章来推广自己的网站。
3. 锚文本 ： 把关键词做一个链接，指向别的网页，这种形式的链接就叫作锚文本。搜索引擎可以根据指向某一个网页的链接的锚文本描述来判断该网页的内容属性。

良好的网站导航和sitemap

网站需要有一个良好的导航，控制根目录和各子目录的关键，通过sitemap(站点地图)可以帮助网站主了解网站结构，也方便搜索引擎收录整个站点。

个性化推荐

HTML5 Geolocation API获得用户的地理位置，进行基于地理位置的运营。