前端的性能优化是一个很宽泛的概念，最终目的都是为了提升用户体验，改善页面性能。面试的时候经常会遇到问谈谈性能优化的手段，这个我分几大部分来概述，具体细节需要自己再针对性的去搜索，只是提供一个索引（太多了写不过来+主要是懒得写）。这里PC端和移动端分开说了，业务场景不同，需要考虑各自的优化手段

目前来看，前端优化的策略有很多，主要包括网络加载，页面渲染，CSS优化，JS执行优化，缓存，图片，协议几大类。这一篇先讲**PC端**的优化策略

网络加载

1. 减少HTTP请求次数

建议尽可能的根据需要去合并静态资源图片、JavaScript代码和CSS文件，减少页面请求数，这样可以缩短页面首次访问的等待时间，另外也要尽量的避免重复资源，防止增加多余的请求

2. 减少HTTP请求大小

除了减少请求资源数，也要减少每个http请求的大小。比如减少没必要的图片，JS，CSS以及HTML等，对文件进行压缩优化，开启GZIP压缩传输内容，缩短网络传输等待延迟

3. 将CSS和JS放到外部文件中，避免使用style和script标签引入

在HTML文件中引入外部的资源可以有效利用浏览器的静态资源缓存。有时候在移动端对请求数比较在意的会为了减少请求把CSS和JS文件直接写到HTML里边，具体根据CSS和JS文件大小和业务场景来分析。如果CSS和JS文件内容较多，逻辑比较复杂，建议放到外部引入 `<link href="https://cdn.bootcss.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">` `<script src="https://cdn.bootcss.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>`

4. 避免页面中空的href和src

当link标签的href属性为空，或者script、img、iframe标签的src属性为空的时候，浏览器在渲染的过程中还是会把href和src的空内容进行加载，直到加载失败。这样就阻塞了页面中其他资源的下载进程，并且最后加载的内容是无效的，因此要尽量避免。

```
<!-- 不推荐 -->
<img src="" alt="占位图"/>
```

5. 为HTML指定Cache-Control或者Expires

为HTML指定Cache-Control或者Expires可以将HTML内容缓存起来，避免频繁向服务器发送请求。在页面Cache-Control或Expires头部又消失，浏览器会直接从缓存读取内容，不向服务器发送请求

```
<meta http-equiv="Cache-Control" content="max-age=7200" />
<meta http-equiv="expires" content="Wed, 20 Jun 2017 22:33:00 GMT" >
```

6. 合理设置Etag和Last-Modified

对于未修改的文件，静态资源服务器会向浏览器端返回304，让浏览器从缓存中读取文件，减少下载的带宽消耗并能减少服务器的负载 `<meta http-equiv="last-modified" content="Mon, 03 Jan 2017 17:45:57 GMT">`

7. 减少页面重定向

一次重定向大概600毫秒的时间开销，为了保证用户能尽快看到页面内容，尽量避免页面的重定向

8. 静态资源不同域名存放

浏览器在同一时刻向同一个域名请求文件的并行下载数是有限的，因此可以理由多个域名的主机来存放不同的静态资源，增大页面加载时资源的并行下载数。

9. 使用静态资源CDN来存储文件

详情搜索 `CDN空间`

10. 使用CDN Combo下载传输内容

CDN的combo技术能把多个资源文件合并引用，减少请求次数。这样可以减少浏览器HTTP请求数，加快资源下载速度。比如淘宝的写法：

```
<link rel="stylesheet" href="//g.alicdn.com/msui/sm/0.6.2/css/??sm.min.css,sm-extend.min.css">
<script type='text/javascript' src='//g.alicdn.com/msui/sm/0.6.2/js/??sm.min.js,sm-
extend.min.js' charset='utf-8'></script>
```

参考: http://www.cnblogs.com/zhengyun_ustc/archive/2012/07/18/combo.html

11. 使用可缓存的AJAX

对于内容相同的请求，有时候没必要每次都从服务器拉取，合理的使用ajax缓存能加快ajax响应速度并减少服务器的压力

```
$.ajax({
    url : url,
    dataType : "json",
    cache: true,
    success: function(data){ // to do something... }
});
```

12. 使用get请求

POST请求会首先发送文件头，然后发送HTTP正文的数据。而使用GET只发送头部，所以在拉取数据时使用GET请求效率更高

13. 减少Cookie的大小并进行Cookie隔离

HTTP请求默认是会带上浏览器端的Cookie一起发送给服务器端的，所以在非必要的情况下要尽量减少Cookie。对于静态的资源，尽量使用不同的域名存放，因为Cookie默认也是不能跨域的，这就做到了不同域名下静态资源请求的Cookie隔离

14. 减少favicon.ico 并缓存

一般一个web应用的favicon.ico是很少改变的，。有利于favicon.ico的重复加载

15. 异步的加载JavaScript资源

异步的JavaScript资源不会阻塞文档解析，所以浏览器会优先渲染页面，延迟加载脚本执行。

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

16. 消除阻塞页面的CSS和JS

对于页面中加载时间过长的CSS或JS文件，需要进行合理的拆分或者延后加载，保证关键的资源能快速加载完成

17. 避免使用CSS import 引用加载CSS

18. 使用prefetch来完成网站预加载

让浏览器预先加载用户访问当前页后极有可能访问的其他资源(页面，图片，视频等)，从而让用户有更好的体验

19. 按需加载

这个跟第二条差不多，特别做单页应用的时候要注意（移动端部分会着重说明）

页面渲染类

1. 把CSS资源引用放到HTML文件顶部

这样浏览器可以优先下载CSS并尽快完成页面渲染

2. JavaScript文件引用放到HTML文件底部

可以防止JavaScript的加载和解析执行对页面渲染造成阻塞。由于JavaScript资源默认是解析阻塞的，除非被标记为异步或者通过其他的方式异步加载，否则会阻塞HTML DOM解析和CSS渲染过程

3. 不要在HTML中直接缩放图片

在HTML中直接缩放图片会导致页面内容的重排重绘，此时可能会使页面中的其他操作产生卡顿，因此要尽量减少在页面中直接进行图片缩放

4. 减少DOM元素数量和深度

HTML中标签元素约的，标签的层级越深，浏览器解析DOM并绘制到浏览器中说花的时间就越长。

5. 尽量避免使用table、iframe等慢元素

内容的渲染是讲table的DOM渲染树全部生成完并一次性绘制到页面上，所以在长表格渲染时很耗性能，应该尽量避免使用，可以考虑用ul代替。尽量使用异步的方式动态的加载iframe，因为iframe内资源的下载进程会阻塞父页面静态资源的下载以及HTML DOM的解析

6. 避免运行耗时的JavaScript

长时间运行的JavaScript会阻塞浏览器构建DOM树、DOM渲染树、渲染页面。所以任何与页面初次渲染无关的逻辑功能都应该延迟加载执行，这和JavaScript资源的异步加载思路一致

7. 避免使用CSS表达式和CSS滤镜

CSS表达式和滤镜的解析渲染速度是很慢的，再有其他解决方案的情况下应该尽量避免使用

```
// 不推荐
.opacity{
   filter: progid:DXImageTransform.Microsoft.Blur(PixelRadius=10, MakeShadow=false)
}
```

至此，PC部分的性能优化点介绍完了。有一些没有讲到的诸如DNS预解析，离线缓存，HTTP2协议，GPU加速等，想着移动端的优化更细，这些内容放到移动端再讲会好一点。因为PC端由于兼容性的问题，很多的优化策略也不能很好的向下降级。尽管列举了很多，但还有少部分遗漏的，欢迎大家补充。前端优化不是一件简简单单的事情，其涉及的内容很多，大家可以根据实际情况将这些方法应用到自己的项目当中去。

关于前端的性能优化，每次提到这个词大家都有很多idea。现在静下来思考下我们用到的各种手段最终可以归纳为三步

一，关键资源字节数

字节数也就是我通常说的减少资源文件（js,css,image,video...）的大小

1，压缩

- 前端使用uglify混淆压缩
- 后端开启gzip
- 对图片进行压缩，使用压缩比例更高的格式（webP）

2，缓存

- 强缓存（http状态码：200），不用请求服务器直接使用本地缓存
- 协商缓存（http状态码：304），使用时先请求服务器若被告知缓存没过期则使用本地缓存，不用下载资源
- 使用localstorage对数据进行存储

3，针对首屏优化

对非关键资源延迟加载、异步加载，减少首屏资源大小

二，关键资源连接数

1，合并请求

- 使用http2.0的多路复用合并请求
- 配置combo，在无法使用http2.0的情况下作为一种合并资源请求的手段

2，减少图片请求数

- 使用spite图
- 使用svg-symbol

3，针对一些场景采用css、js内联的方式

4，使用强缓存减少了一次服务器请求

5，非关键资源延迟、异步加载，减少了首屏资源连接数

三，关键渲染路径

网上有张关于页面渲染路径的图，这里我就不放了，大家有兴趣自己百度下

1，bigpipe分块输出

这里主要是因为要完成一整个页面的输出后端需要处理很多个任务，我们可以将这些多个任务进行分块，谁先完成谁就先输出，最终通过JS回填的方式输出DOM节点。这种方式主要解决了直出页面阻塞的问题

2，bigrender分块渲染

常规的手段就是采用前端模板渲染页面，针对首屏时间主要减少了首次构建DOM树时的节点数

3，针对reflow，repaint，composit路径处理

4，涉及到动画时关于layer的概念render layer、graphics layer

5，css放在头部、js放底部避免阻塞DOM树的构建，

关于css、js的位置对于页面渲染的影响大家可以关注下相关的文章。 核心：css资源不会阻塞DOM树的构建但会阻塞DOM的渲染，JS会阻塞DOM树的构建，CSS会阻塞JS的执行

总结

上面针对性能优化的三步给出了相应的解决方案，这并不是全部，以后大家在思考前端性能优化师可以从这三个基准方向出发