New issue

Jump to bottom

【FE】这么多前端优化点你都记得住吗？ #1

Open zwwill opened this issue on 10 Nov 2017 · 2 comments

Labels 前端 性能优化

Milestone Y-2017

 **zwwill** commented on 10 Nov 2017

> 围绕前端的性能多如牛毛，涉及到方方面面，以我我们将围绕PC浏览器和移动端浏览器的优化策略进行罗列
> 注意，是罗列不是展开，遇到不会不懂的点还请站外扩展

开车速度有点快，坐稳了。

> tips : 这么多前端优化点你都记得住吗？反正我是收藏起来备查的。

PC浏览器前端优化策略

PC端优化的策略很多，如 YSlow（YSlow 是 Yahoo 发布的一款 Firefox 插件，现 Chrome 也可安装，可以对网站的页面性能进行分析，提出对该页面性能优化的建议）原则，或者 Chrome 自带的 Audits 等，总结起来主要包括网络加载类、页面渲染类、CSS 优化类、JavaScript 执行类、缓存类、图片类、架构协议类等几类，下面逐一介绍。

网络加载类

1. 减少 HTTP 资源请求次数

在前端页面中，通常建议尽可能合并静态资源图片、JavaScript 或 CSS 代码，减少页面请求数和资源请求消耗，这样可以缩短页面首次访问的用户等待时间。通过构建工具合并雪碧图、CSS、JavaScript 文件等都是为了减少 HTTP 资源请求次数。另外也要尽量避免重复的资源，防止增加多余请求。

2．减小 HTTP 请求大小

除了减少 HTTP 资源请求次数，也要尽量减小每个 HTTP 请求的大小。如减少没必要的图片、JavaScript、CSS 及 HTML 代码，对文件进行压缩优化，或者使用 gzip 压缩传输内容等都可以用来减小文件大小，缩短网络传输等待时延。前面我们使用构建工具来压缩静态图片资源以及移除代码中的注释并压缩，目的都是为了减小 HTTP 请求的大小。

3．将 CSS 或 JavaScript 放到外部文件中，避免使用 `<style>` 或 `<script>` 标签直接引入

在 HTML 文件中引用外部资源可以有效利用浏览器的静态资源缓存，但有时候在移动端页面 CSS 或 JavaScript 比较简单的情况下为了减少请求，也会将 CSS 或 JavaScript 直接写到 HTML 里面，具体要根据 CSS 或 JavaScript 文件的大小和业务的场景来分析。如果 CSS 或 JavaScript 文件内容较多，业务逻辑较复杂，建议放到外部文件引入。

```
<link rel="stylesheet" href="//cdn.domain.com/path/main.css" >
...
<script src="//cdn.domain.com/path/main.js"></script>
```

4．避免页面中空的 href 和 src

当 `<link>` 标签的 href 属性为空，或 `<script>`、`<img>`、`<iframe>` 标签的 src 属性为空时，浏览器在渲染的过程中仍会将 href 属性或 src 属性中的空内容进行加载，直至加载失败，这样就阻塞了页面中其他资源的下载进程，而且最终加载到的内容是无效的，因此要尽量避免。

```
<!--不推荐-->
<img src="" alt="photo" >
<a href="">点击链接</a>
```

5．为 HTML 指定 Cache-Control 或 Expires

为 HTML 内容设置 Cache-Control 或 Expires 可以将 HTML 内容缓存起来，避免频繁向服务器端发送请求。前面讲到，在页面 Cache-Control 或 Expires 头部有效时，浏览器将直接从缓存中读取内容，不向服务器端发送请求。

```
<meta http-equiv="Cache-Control" content="max-age=7200">
<meta http-equiv="Expires" content="Mon,20Jul201623:00:00GMT">
```

6. 合理设置 Etag 和 Last-Modified

合理设置 Etag 和 Last-Modified 使用浏览器缓存，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 Web 资源下载的带宽消耗并降低服务器负载。

```
<meta http-equiv="last-modified" content="Sun,05 Nov 2017
13:45:57 GMT">
```

7. 减少页面重定向

页面每次重定向都会延长页面内容返回的等待延时，一次重定向大约需要200毫秒不等的时间开销（无缓存），为了保证用户尽快看到页面内容，要尽量避免页面重定向。

8. 使用静态资源分域存放来增加下载并行数

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，因此可以利用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，缩短页面资源加载的时间。通常根据多个域名来分别存储 JavaScript、CSS 和图片文件。

```
<link rel="stylesheet" href="//cdn1.domain.com/path/main.css" >
...
<script src="//cdn2.domain.com/path/main.js"></script>
```

9. 使用静态资源 CDN 来存储文件

如果条件允许，可以利用 CDN 网络加快同一个地理区域内重复静态资源文件的响应下载速度，缩短资源请求时间。

10. 使用 CDN Combo 下载传输内容

CDN Combo 是在 CDN 服务器端将多个文件请求打包成一个文件的形式来返回的技术，这样可以实现 HTTP 连接传输的一次性复用，减少浏览器的 HTTP 请求数，加快资源下载速度。例如同一个域名 CDN 服务器上的 a.js，b.js，c.js 就可以按如下方式在一个请求中下载。

```
<script src="//cdn.domain.com/path/a.js,b.js,c.js"></script>
```

11. 使用可缓存的 AJAX

对于返回内容相同的请求，没必要每次都直接从服务端拉取，合理使用 AJAX 缓存能加快 AJAX 响应速度并减轻服务器压力。

```
$.ajax({
    url : url,
    type : 'get',
    cache : true, //推荐使用缓存
    data : {},
    success (){//...},
    error (){//...}
});
```

12. 使用 GET 来完成 AJAX 请求

使用 XMLHttpRequest 时，浏览器中的 POST 方法会发起两次 TCP 数据包传输，首先发送文件头，然后发送 HTTP 正文数据。而使用 GET 时只发送头部，所以在拉取服务端数据时使用 GET 请求效率更高。

```
$.ajax({
    url : url,
    type : 'get', //推荐使用get完成请求
    data : {},
    success (){//...},
    error(){//...}
});
```

13. 减少 Cookie 的大小并进行 Cookie 隔离

HTTP 请求通常默认带上浏览器端的 Cookie 一起发送给服务器，所以在非必要的情况下，要尽量减少 Cookie 来减小 HTTP 请求的大小。对于静态资源，尽量使用不同的域名来存放，因为 Cookie 默认是不能跨域的，这样就做到了不同域名下静态资源请求的 Cookie 隔离。

14. 缩小 favicon.ico 并缓存

有利于 favicon.ico 的重复加载，因为一般一个 Web 应用的 favicon.ico 是很少改变的。

15. 推荐使用异步 JavaScript 资源

异步的 JavaScript 资源不会阻塞文档解析，所以允许在浏览器中优先渲染页面，延后加载脚本执行。例如 JavaScript 的引用可以如下设置，也可以使用模块化加载机制来实现。

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

使用 async 时，加载和渲染后续文档元素的过程和 main.js 的加载与执行是并行的。使用 defer 时，加载后续文档元素的过程和 main.js 的加载是并行的，但是 main.js 的执行要在页面所有元素解析完成之后才开始执行。

16. 消除阻塞渲染的 CSS 及 JavaScript

对于页面中加载时间过长的 CSS 或 JavaScript 文件，需要进行合理拆分或延后加载，保证关键路径的资源能快速加载完成。

17. 避免使用 CSS import 引用加载 CSS

CSS 中的 `@import` 可以从另一个样式文件中引入样式，但应该避免这种用法，因为这样会增加 CSS 资源加载的关键路径长度，带有 `@import` 的 CSS 样式需要在 CSS 文件串行解析到 `@import` 时才会加载另外的 CSS 文件，大大延后 CSS 渲染完成的时间。

```
<!--不推荐-->
<style>
    @import "path/main.css";
</style>
```

```
<!--推荐-->
<link rel="stylesheet" href="//cdn1.domain.com/path/main.css" >
```

页面渲染类

1. 把 CSS 资源引用放到 HTML 文件顶部

一般推荐将所有 CSS 资源尽早指定在 HTML 文档 `<head>` 中，这样浏览器可以优先下载 CSS 并尽早完成页面渲染。

2. JavaScript 资源引用放到 HTML 文件底部

JavaScript 资源放到 HTML 文档底部可以防止 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异步方式加载，否则会阻塞 HTML DOM 解析和 CSS 渲染的过程。

3. 尽量预先设定图片等大小

在加载大量的图片元素时，尽量预先限定图片的尺寸大小，否则在图片加载过程中会更新图片的排版信息，产生大量的重排

4. 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。

5. 减少 DOM 元素数量和深度

HTML 中标签元素越多，标签的层级越深，浏览器解析 DOM 并绘制到浏览器中所花的时间就越长，所以应尽可能保持 DOM 元素简洁和层级较少。

```
<!--不推荐-->
<div>
    <span>
        <a href="javascript:void(0);">
            <img src="./path/photo.jpg" alt="图片">
        </a>
    </span>
</div>

<!--推荐-->
```

```
<!--推荐-->
<img src="./path/photo.jpg" alt="图片" >
```

6. 尽量避免在选择器末尾添加通配符

CSS 解析匹配到 渲染树的过程是从右到左的逆向匹配，在选择器末尾添加通配符至少会增加一倍多计算量。

7. 减少使用关系型样式表的写法

直接使用唯一的类名即可最大限度的提升渲染引擎绘制渲染树等效率

8. 尽量减少使用JS动画

JS 直接操作 DOM 极容易引起页面的重排

9. CSS 动画使用 translate、scale 代替 top、height

尽量使用 CSS3 的 translate、scale 属性代替 top、left 和 height、width，避免大量的重排计算

10. 尽量避免使用 `<table>` 、 `<iframe>`

`<table>` 内容的渲染是将 table 的 DOM 渲染树全部生成完并一次性绘制到页面上的，所以在长表格渲染时很耗性能，应该尽量避免使用它，可以考虑使用列表元素 `<ul>` 代替。尽量使用异步的方式动态添加 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与 CSS 及 HTML DOM 的解析。

11. 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是一致的。

12. 避免使用 CSS 表达式或 CSS 滤镜

CSS 表达式或 CSS 滤镜的解析渲染速度是比较慢的，在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.Alpha( opacity
= 50 );
}
```

移动端浏览器前端优化策略

相对于桌面端浏览器，移动端 Web 浏览器上有一些较为明显的特点：设备屏幕较小、新特性兼容性较好、支持一些较新的 HTML5 和 CSS3 特性、需要与 Native 应用交互等。但移动端浏览器可用的 CPU 计算资源和网络资源极为有限，因此要做好移动端 Web 上的优化往往需要做更多的事情。首先，在移动端 Web 的前端页面渲染中，桌面浏览器端上的优化规则同样适用，此外针对移动端也要做一些极致的优化来达到更好的效果。需要注意的是，并不是移动端的优化原则在桌面浏览器端就不适用，而是由于兼容性和差异性的原因，一些优化原则在移动端更具代表性。

网络加载类

1. 首屏数据请求提前，避免 JavaScript 文件加载后才请求数据

为了进一步提升页面加载速度，可以考虑将页面的数据请求尽可能提前，避免在 JavaScript 加载完成后才去请求数据。通常数据请求是页面内容渲染中关键路径最长的部分，而且不能并行，所以如果能将数据请求提前，可以极大程度上缩短页面内容的渲染完成时间。

2. 首屏加载和按需加载，非首屏内容滚屏加载，保证首屏内容最小化

由于移动端网络速度相对较慢，网络资源有限，因此为了尽快完成页面内容的加载，需要保证首屏加载资源最小化，非首屏内容使用滚动的方式异步加载。一般推荐移动端页面首屏数据展示延时最长不超过3秒。目前中国联通 3G 的网络速度为 338KB/s（2.71Mb/s），所以推荐首屏所有资源大小不超过 1014KB，即大约不超过 1MB。

3. 模块化资源并行下载

在移动端资源加载中，尽量保证 JavaScript 资源并行加载，主要指的是模块化 JavaScript 资源的异步加载，例如AMD的异步模块，使用并行的加载方式能够缩短多个文件资源的加载时间。

4．inline 首屏必备的 CSS 和 JavaScript

通常为了在 HTML 加载完成时能使浏览器中有基本的样式，需要将页面渲染时必备的 CSS 和 JavaScript 通过 `<script>` 或 `<style>` 内联到页面中，避免页面 HTML 载入完成到页面内容展示这段过程中页面出现空白。

```
<!DOCTYPE html>
<html lang="en">
    <head>
    <meta charset="UTF-8">
    <title>样例</title>
    <meta name="viewport" content="width=device-width,minimum-
scale=1.0,maximum-scale=1.0,user-scalable=no">
    <style>
    /*必备的首屏CSS*/
    html,body{
        margin:0;
        padding:0;
        background-color:#ccc;
    }
    </style>
</head>
<body>
</body>
</html>
```

5．meta dns prefetch 设置 DNS 预解析

设置文件资源的 DNS 预解析，让浏览器提前解析获取静态资源的主机 IP，避免等到请求时才发起 DNS 解析请求。通常在移动端 HTML 中可以采用如下方式完成。

```
<!--cdn域名预解析-->
<meta http-equiv="x-dns-prefetch-control" content="on" >
<link rel="dns-prefetch" href="//cdn.domain.com" >
```

6．资源预加载

对于移动端首屏加载后可能会被使用的资源，需要在首屏完成加载后尽快进行加载，保证在用户需要浏览时已经加载完成，这时候如果再去异步请求就显得很慢。

7．合理利用MTU策略

通常情况下，我们认为 TCP 网络传输的最大传输单元（Maximum Transmission Unit，MTU）为 1500B，即一个RTT（Round-Trip Time，网络请求往返时间）内可以传输的数据量最大为 1500 字节。因此，在前后端分离的开发模式中，尽量保证页面的 HTML 内容在 1KB 以内，这样整个 HTML 的内容请求就可以在一个 RTT 内请求完成，最大限度地提高 HTML 载入速度。

缓存类

1．合理利用浏览器缓存

除了上面说到的使用 Cache-Control、Expires、Etag 和 Last-Modified 来设置 HTTP 缓存外，在移动端还可以使用 localStorage 等来保存 AJAX 返回的数据，或者使用 localStorage 保存 CSS 或 JavaScript 静态资源内容，实现移动端的离线应用，尽可能减少网络请求，保证静态资源内容的快速加载。

2．静态资源离线方案

对于移动端或 Hybrid 应用，可以设置离线文件或离线包机制让静态资源请求从本地读取，加快资源载入速度，并实现离线更新。关于这块内容，我们会在后面的章节中重点讲解。

3．尝试使用 AMP HTML

AMP HTML 可以作为优化前端页面性能的一个解决方案，使用 AMP Component 中的元素来代替原始的页面元素进行直接渲染。

```
<!--不推荐-->
<video width="400" height="300"
src="http://www.domain.com/videos/myvideo.mp4"
poster="path/poster.jpg">
    <div fallback>
        <p>Your browser doesn’t support HTML5 video</p>
    </div>
    <source type="video/mp4" src="foo.mp4">
    <source type="video/webm" src="foo.webm">
</video>

<!--推荐-->
<amp-video width="400" height="300"
src="http://www.domain.com/videos/myvideo.mp4"
poster="path/poster.jpg">
```

```
    <div fallback>
        <p>Your browser doesn’t support HTML5 video</p>
    </div>
    <source type="video/mp4" src="foo.mp4">
    <source type="video/webm" src="foo.webm">
</amp-video>
```

4．尝试使用 PWA 模式

PWA（Progressive Web Apps）是 Google 提出的用前沿的 Web 技术为网页提供 App 般使用体验的一系列方案。

图片类

1．图片压缩处理

在移动端，通常要保证页面中一切用到的图片都是经过压缩优化处理的，而不是以原图的形式直接使用的，因为那样很消耗流量，而且加载时间更长。

2．使用较小的图片，合理使用 base64 内嵌图片

在页面使用的背景图片不多且较小的情况下，可以将图片转化成 base64 编码嵌入到 HTML 页面或 CSS 文件中，这样可以减少页面的 HTTP 请求数。需要注意的是，要保证图片较小，一般图片大小超过 2KB 就不推荐使用 base64 嵌入显示了。

```
.class-name{
    background-image :
url('

}
```

3．使用更高压缩比格式的图片

使用具有较高压缩比格式的图片，如 webp（需要设计降级兼容方案）等。在同等图片画质的情况下，高压缩比格式的图片体积更小，能够更快完成文件传输，节省网络流量。

```
<img src="//cdn.domain.com/path/photo.webp" alt="webp格式图片" >
```

4．图片懒加载

为了保证页面内容的最小化，加速页面的渲染，尽可能节省移动端网络流量，页面中的图片资源推荐使用懒加载实现，在页面滚动时动态载入图片。

```
<img data-src="//cdn.domain.com/path/photo.jpg" alt="懒加载图片" >
```

5．使用 MediaQuery 或 srcset 根据不同屏幕加载不同大小图片

在介绍响应式的章节中我们了解到，针对不同的移动端屏幕尺寸和分辨率，输出不同大小的图片或背景图能保证在用户体验不降低的前提下节省网络流量，加快部分机型的图片加载速度，这在移动端非常值得推荐。

6．使用 iconfont 代替图片图标

在页面中尽可能使用 iconfont 来代替图片图标，这样做的好处有以下几个：

- 使用 iconfont 体积较小，而且是矢量图，因此缩放时不会失真；
- 可以方便地修改图片大小尺寸和呈现颜色。

但是需要注意的是，iconfont 引用不同 webfont 格式时的兼容性写法，根据经验推荐尽量按照以下顺序书写，否则不容易兼容到所有的浏览器上。

```
@font-face{
    font-family:iconfont;
    src:url("./iconfont.eot");
    src:url("./iconfont.eot?#iefix")  format("eot"),
        url("./iconfont.woff")  format("woff"),
        url("./iconfont.ttf")  format("truetype");
}
```

7．定义图片大小限制

加载的单张图片一般建议不超过 30KB，避免大图片加载时间长而阻塞页面其他资源的下载，因此推荐在 10KB 以内。如果用户上传的图片过大，建议设置告警系统，帮助我们观察了解整个网站的图片流量情况，做出进一步的改善。

8．强缓存策略

对于一些「永远」不会变的图片可以使用强缓存的方式缓存在用户的浏览器上。

脚本类

1. 尽量使用 id

选择器选择页面 DOM 元素时尽量使用 id 选择器，因为 id 选择器速度最快。

2. 合理缓存 DOM 对象

对于需要重复使用的 DOM 对象，要优先设置缓存变量，避免每次使用时都要从整个DOM树中重新查找。

```
//不推荐
$('#mod.active').remove('active');
$('#mod.not-active').addClass('active');

//推荐
let $mod=$('#mod');
$mod.find('.active').remove('active');
$mod.find('.not-active').addClass('active');
```

3. 页面元素尽量使用事件代理，避免直接事件绑定

使用事件代理可以避免对每个元素都进行绑定，并且可以避免出现内存泄露及需要动态添加元素的事件绑定问题，所以尽量不要直接使用事件绑定。

```
//不推荐
$('.btn').on('click',function(e){
    console.log(this);
});

//推荐
$('body').on('click','.btn',function(e){
    console.log(this);
});
```

4. 使用 touchstart 代替 click

由于移动端屏幕的设计， touchstart 事件和 click 事件触发时间之间存在 300 毫秒的延时，所以在页面中没有实现 touchmove 滚动处理的情况下，可以使用 touchstart 事件来代替元素的 click 事件，加快页面点击的响应速度，提高用户体验。但同时我们也要注意页面重叠元素 touch 动作的点击穿透问题。

```
//不推荐
$('body').on('click','.btn',function(e){
    console.log(this);
});

//推荐
$('body').on('touchstart','.btn',function(e){
    console.log(this);
});
```

5．避免 touchmove、scroll 连续事件处理

需要对 touchmove、scroll 这类可能连续触发回调的事件设置事件节流，例如设置每隔 16ms（60 帧的帧间隔为 16.7ms，因此可以合理地设置为 16ms ）才进行一次事件处理，避免频繁的事件调用导致移动端页面卡顿。

```
//不推荐
$('.scroller').on('touchmove','.btn',function(e){
    console.log(this);
});

//推荐
$('.scroller').on('touchmove','.btn',function(e){
    let self=this;
    setTimeout(function(){
        console.log(self);
    },16);
});
```

6．避免使用 eval、with，使用 join 代替连接符＋，推荐使用 ECMAScript6 的字符串模板

这些都是一些基础的安全脚本编写问题，尽可能使用较高效率的特性来完成这些操作，避免不规范或不安全的写法。

7．尽量使用 ECMAScript6＋的特性来编程

ECMAScript6＋ 一定程度上更加安全高效，而且部分特性执行速度更快，也是未来规范的需要，所以推荐使用 ECMAScript6＋ 的新特性来完成后面的开发。

渲染类

1. 使用 Viewport 固定屏幕渲染，可以加速页面渲染内容

一般认为，在移动端设置 Viewport 可以加速页面的渲染，同时可以避免缩放导致页面重排重绘。在移动端固定 Viewport 设置的方法如下。

```
<!--设置viewport不缩放-->
<meta name="viewport" content="width=device-width,initial-
scale=1.0,maximum-scale=1.0,user-scalable=no">
```

2. 避免各种形式重排重绘

页面的重排重绘很耗性能，所以一定要尽可能减少页面的重排重绘，例如页面图片大小变化、元素位置变化等这些情况都会导致重排重绘。

3. 使用 CSS3 动画，开启GPU加速

使用 CSS3 动画时可以设置 transform:translateZ(0) 来开启移动设备浏览器的GPU图形处理加速，让动画过程更加流畅，但需要注意的是，在 Native WebView 下 GPU 加速有几率产生 App Crash。

```
-webkit-transform:translateZ(0);
    -ms-transform:translateZ(0);
     -o-transform:translateZ(0);
        transform:translateZ(0);
```

4. 合理使用 Canvas 和 requestAnimationFrame

选择 Canvas 或 requestAnimationFrame 等更高效的动画实现方式，尽量避免使用 setTimeout、setInterval 等方式来直接处理连续动画。

5. SVG 代替图片

部分情况下可以考虑使用 SVG 代替图片实现动画，因为使用 SVG 格式内容更小，而且 SVG DOM 结构方便调整。

6．不滥用 float

在 DOM 渲染树生成后的布局渲染阶段，使用 float 的元素布局计算比较耗性能，所以尽量减少 float 的使用，推荐使用固定布局或 flex-box 弹性布局的方式来实现页面元素布局。

7．不滥用 web 字体或过多 font-size 声明

过多的 font-size 声明会增加字体的大小计算，而且也没有必要的。

8．做好脚本容错

脚本容错可以避免「非正常环境」的执行错误影响页面的加载和不相关功能的使用

架构协议类

1．尝试使用 SPDY 和 HTTP2

在条件允许的情况下可以考虑使用 SPDY 协议来进行文件资源传输，利用连接复用加快传输过程，缩短资源加载时间。HTTP2 在未来也是可以考虑尝试的。

2．使用后端数据渲染

使用后端数据渲染的方式可以加快页面内容的渲染展示，避免空白页面的出现，同时可以解决移动端页面SEO的问题。如果条件允许，后端数据渲染是一个很不错的实践思路。后面的章节会详细介绍后端数据渲染的相关内容。

3．使用 NativeView 代替 DOM 的性能劣势

可以尝试使用 NativeView 的 MNV＊ 开发模式来避免 HTML DOM 性能慢的问题，目前使用 MNV＊ 的开发模式已经可以将页面内容渲染体验做到接近客户端 Native 应用的体验了。但需要避免 js Framework 和 native Framework 的频繁交互。

总结

关于页面优化的常用技术手段和思路主要包括以上这些，尽管列举出很多，但仍可能有少数遗漏，可见前端性能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，要想全部做到几乎是不可能的，但做到用户可接受的原则还是很容易实现的。

另外，如果你有比较好的优化点想要扩充，欢迎下方评论。

+😀

 **zwwill** added 前端 性能优化 labels on 10 Nov 2017

 **zwwill** assigned **zwwill** and unassigned **zwwill** on 10 Nov 2017

 **zwwill** modified the milestone: **Y-2017** on 10 Nov 2017

 **zwwill** changed the title ~~这么多前端优化点你都记得住吗?~~ 【FE】这么多前端优化点你都记得住吗? on 1 Dec 2017

 **zwwill** referenced this issue on 1 Dec 2017

【布局】CSS布局方案 #14 Open

 **xwchris** referenced this issue on 3 Sep 2018

技术备忘-优秀的文章及工具 #61 Closed

 **guoshuai93** referenced this issue on 11 Oct 2018

【转】这么多前端优化点你都记得住吗? #37 Open

 **zwwill** referenced this issue on 30 Oct 2018

【布局】聊聊为什么淘宝要提出「双飞翼」布局 #11 Open

zayfen commented on 24 Dec 2018

`//推荐 $('.scroller').on('touchmove','.btn',function(e){ let self=this; setTimeout(function(){ console.log(self); },16); });`
在下次touchmove执行之前是要先清楚之前的timeout吧,
下面这样写会不会好很多呢
`

```
//推荐
var timeoutHandler = -1;
$('.scroller').on('touchmove','.btn',function(e){
let self=this;
clearTimeout(timeoutHandler)
timeoutHandler = setTimeout(function(){
console.log(self);
},16);
});
```

`

+😀

DaiZhaoedu commented on 8 Jul • edited

> CSS 动画使用 translate、scale 代替 top、height

这条优化没看懂。
我的理解是 用translate 的x,y轴代替 position 的top， height吗?

+😀

Assignees

No one assigned

Labels

前端

性能优化

Projects

None yet

Milestone

Y-2017

3 participants