PERFORMANCE

1. 前言

性能判定标准 + 搭建性能优化指标采集环境 + 性能优化解决方案

性能优化工具

2. 性能关注指标

- 加载时间：
  - Time to First Impression: 从用户输入url到网页开始显示title;
  - Time to onLoad Event: 从页面开始显示内容到浏览器开始触发OnLoad().只有当初始的文本和所引用的对象加载完成，浏览器才开始触发Unload()
  - Time to Fully Loaded: 从上一时间段末到整个网页完全加载完成(所有OnLoad()及相关的动态资源加载完成)。PS：含有timeout或定时刷新之类处理时较为难判断结束点
- 资源情况：
  - Total Number of Requests:包括html网页请求 | css | js 资源下载及其它网络请求

    - Total Number of HTTP 300s|400s|500s:避免返回状态为重定向|客户端错误|服务器端错误的http请求，以提高页面load的时间。造成这些状态的原因经常是服务器的实施、配置和部署的问题
    - Total Size of Web Site:构成网页元素总的大小，会对下载时间造成重要的影响
    - Total Size of Images|CSS|JS:这类资源在网页元素大小中占主要比例
    - Total Number of XHR:通过JS异步从服务器获得数据的请求数(一些JS框架提供了跟服务器端的更新机器，就是XHR请求，通过配置可以减少XHR请求的数目)
- 网络连接
    - DNS Time：DNS查询的时间。网页请求会产生一次寻找该网页资源所在主机的DNS查询，在同个域名进行网页切换不会造成新的DNS查询。
    - Connect Time: 指浏览器和服务器之间建立TCP|IP连接的时间，对于SSL连接包括握手的时间。网络连接过慢|使用SSL|使用短连接而非常连接都是造成connect time较多的原因。
    - Server Time：指收到请求后服务器逻辑处理的时间
    - Transfer Time:这一指标与浏览器和服务器之间的连接速度相一致，通过减小传输内容或使用cdn来降低Transfer Time。
    - Wait Time：等待时间和同一个域中服务资源的数量直接相关。每个域的浏览器的物理网络的限制，导致资源等待可用的连接。减少资源的数量，或将资源散布在不同的域，能将这一时间降低。平均等待时间的大小更能反映等待时间是否需要注意
    - Number of Domains | Single Resource Domains:部署网站资源的域主机数量是很重要的，因为它会影响DNS|连接|等待时间。专门用户资源下载的域是必要的，它将直接减少等待时间。应避免单一的资源域，否则你将为DNS查询以及资源下载付出昂贵的代价

3. 网站设计优化点指标

- 点击热力图：根据用户点击的位置，我们可以画出整个页面的点击热力图，可以很直观的了解到页面的热点区域，从而在版本迭代时优化布局及业务流程
- 访问浏览器：统计到浏览器的占比，对于是否继续兼容旧版本浏览器、是否采用新技术（HTML5、CSS3 等）等提供参考价值
- 异常：QA 能够覆盖到大部分的 bug，但肯定也会有一些 bug 在线上出现。JS 的异常捕获只有两种方式：window.onerror、try/catch，关于我们是如何做的将在后续的文章中有详细的描述，这里只列出捕获到异常时，一般需要采集哪些信息（主要用来 debug 异常）：
    - 异常的提示信息：这是识别一个异常的最重要依据，如：’e.src’ 为空或不是对象
    - JS 文件名
    - 异常所在行
    - 发生异常的浏览器
    - 堆栈信息：必要的时候需要函数调用的堆栈信息，但是注意堆栈信息可能会比较

大，需要截取

4. 数据加载期优化技巧

4.1. 图片控制

4.1.1. 图片格式(启用webp)

WebP是一种支持有损压缩和无损压缩的图片文件格式，派生自视频编码格式 VP8。无损压缩后的 WebP 比 PNG 文件少了 26% 的文件大小，有损压缩在具有同等SSIM索引的情况下WebP 比 JPEG 文件少25-34%的文件大小。WebP支持无损透明度（也叫做alpha通道），支持动画格式Animated WebP 。 页面加载时进行v 环境探测, 如页面渲染环境支持WebP就替换页面中的图片链接为WebP格式的版本。

WebP格式转换

4.1.2. 优化高分屏和弱网适配

高像素密度显示屏（高PPI）:在浏览器里一个CSS像素往往能对应两个或更多个设备像素，为了追求最好的显示效果,我们会采用数倍于浏览器CSS像素标识的图片尺寸. 但是如果你采用了2x (两倍CSS像素分辨率) 的图片,由于水平和垂直像素都进行了加倍，最终图片体积会增加4倍(内存占用也会增加4倍). 同理,如果你采用了3x的图片，最终增加的传输体积会增至9倍.

最多使用2x(两倍CSS像素分辨率)的图片。 如果页面专门用于可控的客户端内，我们会根据从客户端获取的网络情况替换页面所使用的图片资源. 在最糟糕的网络环境(2G移动网络)，我们甚至会采用按30%质量进行压缩的图片以替换原始图片链接。

4.1.3. 单张图片大小限制

针对图片设置单张图片大小不超过50Kb的限制。在每次发布时,会对页面图片进行检查, 如果图片超过这个限制仍需要发布，请走特别的申请流程.

4.1.4. 使用CSS3/SVG/ICON Font替代图片

每使用Sprite技术展示一个图标，都需要浏览器把整个大图解码到内存中一次，占用的内存和解码时间往往是可观的。

图标使用CSS3,SVG和ICON Font技术，在任何分辨率和缩放设置都可以提供清晰的效果并减小传输和内存的开销.

4.1.5. 图像的 BASE64 编码

图片的下载不用向服务器发出请求，而可以随着 HTML 的下载同时下载到本地.将图片数据编码成 BASE64 的字符串，使用该字符串代替图像地址。假设用 S代表这个 BASE64 字符串，那么就可以使用 `<img src="">` 来显示这个图像。适用的情况是浏览器连接服务器的时间 > 图片下载时间，也就是发起连接的代价要大于图片下载，那么这个时候将图片编码为 BASE64 字符串，就可以避免连接的建立，提高效率。如果图片较大的话，使用 BASE64 编码虽然可以避免连接建立，但是相对于图像下载，请求的建立只占很小的比例，如果用 BASE64，对于动态网页来说图像缓存就会失效（静态网页可以缓存），而且 BASE64 字符串的总大小要大于纯图片的大小

使用webpack等构建工具或者Base64 Online

4.1.6. 懒加载和预加载

预加载和懒加载，是一种改善用户体验的策略，它实际上并不能提高程序性能，但是却可以明显改善用户体验或减轻服务器压力。

- 预加载原理:在用户查看一张图片时，就将下一张图片先下载到本地，而当用户真正访问下一张图片时，由于本地缓存的原因，无需从服务器端下载，从而达到提高用户体验的目的。

```
// 首先定义了 Image 对象，并且声明了需要预加载的图像数组，
// 然后逐一的开始加载（.src=images[i]）。
// 如果已经在缓存里，则不做其他处理；
// 如果不在缓存，监听 onload 事件，它会在图片加载完毕时调用。
function  preload(callback) {
    var imageObj = new Image();
    images = new Array();
    images[0]="pre_image1.jpg";
    images[1]="pre_image2.jpg";
    images[2]="pre_image3.jpg";
    for(var i=0; i<=2; i++) {
        imageObj.src=images[i];
        if (imageObj.complete) { // 如果图片已经存在于浏览器缓存，直接调用回调
函数
            callback.call(imageObj);
        } else {
            imageObj.onload = function () {// 图片下载完毕时异步调用
callback 函数
```

```
            callback.call(imageObj);// 将回调函数的 this 替换为 Image 对
象
        };
    }
}
function callback(){
    alert(this.src + “已经加载完毕 , 可以在这里继续预加载下一组图片”);
}
```

- 懒加载:在用户需要的时候再加载。当一个网页中可能同时有上百张图片，而大部分情况下，用户只看其中的一部分，如果同时显示上百张，则浪费了大量带宽资源，因此可以当用户往下拉动滚动条时，才去请求下载被查看的图像，这个原理与 word 的显示策略非常类似。在 JavaScript 中，它的基本原理是首先要有一个容器对象，容器里面是 img 元素集合。用隐藏或替换等方法，停止 img 的加载，也就是停止它去下载图像。然后历遍 img 元素，当元素在加载范围内，再进行加载（也就是显示或插入 img 标签）。加载范围一般是容器的视框范围，即浏览者的视觉范围内。当容器滚动或大小改变时，再重新历遍元素判断。如此重复，直到所有元素都加载后就完成。当然对于开发来讲，可以选择已有的成熟组件，Vanilla JavaScript Lazy Load Plugin 是基于 JQuery 的懒加载组件。

4.2. 资源大小

详情请见webpack篇

4.3. 网络请求

4.3.1. 域名收敛

若页面中引入的各种资源来自不同的域名，每增加一个域名,都会增加一次域名解析开销。 在复杂的国内移动互联网网络环境下，不同域名的解析速度可能会相差数十倍。另不同浏览器对于同域名资源并发下载数一般2-6个。

规定一个页面所产生的域名解析数不能超过5个。有意识的收敛页面资源所需解析的域名数, 特别是会阻塞页面渲染的CSS,JS,Font资源。

4.3.2. 页面资源请求数

如果是长期维护的产品型页面，页面中引入的静态资源除最通用的基础库外需要按依赖顺序进行合并压缩. 一般是JS和CSS请求各一。

开发了工具把全部的CSS和JS资源内联入页面，从而实现除图片外的其余资源One Request就能获得.另外，资源请求重定向也应该尽量避免,少一次重定向，少一个请求数.

4.3.3. 文本数据的优化与压缩

文本数据(HTML,CSS,JavaScript)的优化与压缩分为三个阶段，分别是发布准备(去除注释,合并CSS声明,去除不会被执行的JS代码块), 编译期压缩(合并文件，去除空格，混淆) 和 传输阶段压缩(GZip) .

关键在于梳理流程确保压缩的自动化和服务器GZip指令被正确配置。

4.3.4. 数据接口优化与监控

前后端分离：在页面加载完成后再通过接口获取JSON数据并在前端进行页面渲染。带来了页面第一次加载速度的提升，但常常把原有的性能问题隐藏了起来。需要花功夫优化的数据获取并最终呈现时间往往被隐藏在空页面快速呈现的表象之下。更严重的情况发生在需要从多个不同接口获取数据，并且这些接口调用还存在互相依赖的情况下，一旦出现这样的情况，页面性能往往是不升反降的.

要求数据在后端组装完成后再交由前端渲染，一屏中完整渲染所需数据不能来自多过一个接口。所有数据源统一收敛到单一的接口服务层，以便进行统计和监控。