Web 前端性能优化方案与实践

王 成 李少元 郑黎晓 缑 锦 曾梅琴 刘慧敏
(华侨大学计算机科学与技术学院 福建 厦门 362021)

摘 要 针对 Web 前端性能低下的问题,通过分析归纳 Web 中从后端到前端的 B/S 架构原理、浏览器缓存、浏览器的加载方式、服务器关于 HTTP 相关的配置等过程中一些影响前端性能优化的因素,系统地提出一个旨在提高网页加载速度、呈现速度和用户体验,整体性、通用性强的完整 Web 前端性能优化解决方案。该解决方案包括服务器端优化、HTML 优化、JavaScript 优化、CSS 优化、图片优化等内容。并在 HTTP 代理工具 Fiddler 搭建的 512 KB 慢网速下通过 SpeedTracer 监测 UI Thread,寻找基于 HTML5 技术的 Web 移动电子商务项目"指尖点餐系统"的点餐页面前端性能中的瓶颈,根据所提出的 Web 前端性能优化解决方案对其进行优化实践。优化前后的 Timeline 以及 UI Thread 对比分析表明,优化后加载时间降低了 82%,页面渲染降低了 32%,脚本执行减少了 79%。

关键词 Web 前端性能优化 服务器端优化 HTML 优化 JavaScript 优化 CSS 优化 图片优化 前端测试环境 TimeLine UI Thread

中图分类号 TP302.7 **文献标识码** A DOI: 10.3969/j.issn.1000-386x.2014.12.021

OPTIMISATION SCHEME FOR WEB FRONT-END PERFORMANCE AND ITS PRACTICE

Wang Cheng Li Shaoyuan Zheng Lixiao Gou Jin Zeng Meiqin Liu Huimin
(*College of Computer Science and Technology, Huaqiao University, Xiamen 362021, Fujian, China*)

Abstract Aiming at the poor performance of Web front-end, we put forward systematically a complete Web front-end performance optimisation solution with strong integrity and universality targeted at increasing webpages loading speed, rendering speed and user experience by analysing and generalising the principles of B/S frameworks from rear-end to front-end in Web, the browser cache, the browser loading means, and the factors that affect the front-end performance optimisation in server configuration in regard to HTTP and other processes. The solution includes server-side optimisation, HTML optimisation, JavaScript optimisation, CSS optimisation, image optimisation and so on. In slow network speed of 512KB built by HTTP proxy tool Fiddler, we monitor the UI Thread through SpeedTracer, and look for the bottleneck of front performance in ordering webpage of "fingertips ordering system", a Web mobile e-commerce project based on HTML5 technology, and carry out the optimisation practice according to the proposed solution. Comparative analyses of Timeline and UI thread before and after the optimisation show that when optimised, the loading time reduces by 82%, the page rendering by 32%, and the script execution by 79%.

Keywords Web front-end performance optimisation Server-side optimisation HTML optimisation JavaScript optimisation CSS optimisation Image optimisation Front-end test environment TimeLine UI Thread

0 引 言

加载时间很大程度上影响着用户体验和网站的竞争力。据美国网上媒体报道,Google 检索到的网页的加载速度平均为 2.45秒[1],而购物网站的平均网页加载速度为 7.25 秒[2]。美国研究生项目资讯网站的调查结果[3]表明,网页加载时间超过 4 秒,约有四分之一的人会放弃打开该网页。调查机构 KissMetrics 研究发现:网页加载速度影响用户消费,如果电子商务每天收入为 10 万美元,那么 1 秒的延迟就会让该网站每年损失 250 万美元[4]。

截止至 2013 年 4 月,互联网站总数近 10 亿,每秒新增 7.9 个新网民。如此高速发展的 Web,却普遍存在前端性能低下的问题[5]。例如:2011 年百度年会推出的百度新首页,后端的平均时间为 60 ms,而前端的平均时间却为 1.3 秒[6]。与此同时,国内的网速普遍比发达国家慢很多,中国大陆网速排在世界第 71 名,平均网速为 1.774 Mbps,远低于世界平均水平[7]。在网速低下的环境下,对 Web 进行前端性能优化显得尤为重要。

如今,Web 的高速发展,以往的以文档为核心的 HTML 和 XHTML 标准已经无法满足 Web 发展的需要。新一代的 HTML5 应用旨在改变这一切,其拥有很多强大的用于交互、多媒体和本地化等各个方面的标签以及应用编程接口[8]。但这些的实现意味着需要更好前端性能,特别是在移动端网速普遍较慢的环境下。

早在 21 世纪初,雅虎、谷歌等知名互联网公司已经开始重视前端性能优化,并提出了很多解决方案以及相关的工具。比

收稿日期:2013-06-02。国家自然科学基金项目(51305142);华侨大学人才引进科研启动项目(12Y0347)。王成,讲师,主研领域:电子商务网站开发与智能计算。李少元,本科。郑黎晓,讲师。缑锦,副教授。曾梅琴,本科生。刘慧敏,本科生。

如谷歌的 PageSpeed、SpeedTracer、雅虎的首席性能工程师 Steve Souders 根据性能优化的经验编写了《High Performance Web Sites》[9] 以及他与另外 8 位 Web 前端专家级特约作者编写的《Even Faster Web Sites》[10] 提供了提升网站性能的最佳实践和实用建议。国内 2010 年,张紫微等人[11] 通过对影响 Web 前端性能的各种因素,包括 HTTP 协议、浏览器工作方式、缓存机制、页面大小、页面结构以及 Ajax 等前端相关理论的分析,有针对性地提出了前端优化的多种技术开展方法,设计并实现了一个基于等待时间提升优先级的优先级请求队列,使所有的异步请求都放入优先级队列,由该队列管理请求和发送请求,解决了 Ajax 浏览器的 2 连接请求问题。高洁璇[12] 采用 Ajax 技术建立了 Ajax 化页面模型,从传统的缓存思想出发,针对热点页面建立了页面缓存模型,使请求冗余进一步减少,降低了服务器负载,利用并行思想,设计了适用于信息复杂、可模块化的页面并行加载模型,降低请求响应过程中的冗余,提高了服务器响应速度,并缩短用户等待时间,获得了更好的用户体验。

在性能诊断与优化处理工具方面,周鹏等[13] 提出了基于 Spirent 的 Web 应用性能评测。聂应高[14] 以湖北省高校数字图书馆门户为例,进行了基于 Page Speed 的网站前端性能优化分析和实践。连志刚[15] 认为 Web 性能测试难度大、周期长、要求高是一直困扰测试人员的主要问题,研究了基于 RBI 的马尔科夫 Web 系统性能测试过程模型。赵佳佳[16] 运用性能测试工具 LoadRunner 等建立场景并从全面分析研究,尽可能真实地模拟大量用户的并发操作等行为,并对测试结果进行完善,利用单因素隔离与对比等方法分析影响 Web 性能的瓶颈。但这些方案往往比较宽泛零散,缺乏系统性,实践对比不足,与项目挂钩太紧密,缺乏通用性性能测试,没形成一个完整的解决方案。

据此,通过分析归纳 Web 中从后端到前端的 B/S 架构原理、浏览器缓存、浏览器的加载方式、服务器关于 HTTP 相关的配置等过程中一些影响前端性能优化的因素,系统地提出了一个旨在提高网页加载速度、呈现速度和用户体验,整体性、通用性强的完整 Web 前端性能优化解决方案,包括服务器端优化、HTML 优化、JavaScript 优化、CSS 优化、图片优化,过程尽量避免项目的特殊性,强调实践的可行性以及通用性。针对一个基于 HTML5 技术的 Web 移动应用“指尖点餐系统”选用适当的性能检测工具搭建测试环境,寻找该系统前端性能优化上的瓶颈,并结合该项目特点,根据所提出的 Web 前端性能优化解决方案对其进行优化实践,对比优化前后的性能。

1 Web 性能优化方案

1.1 Web 性能优化相关理论

(1) B/S 架构

从浏览器对服务器进行页面请求的过程[16] 如下所示:

① 首先用户在浏览器输入网址或者通过其它应用程序请求 url。

② DNS 解析域名,返回该域名所指向的网站 ip 地址。

③ 浏览器对对应的服务器发送 HTTP 请求。

④ 服务器接收到浏览器的发送的 HTTP 请求。

⑤ 服务器解悉浏览器请求的 URL,根据 URL 确定请求的目标资源文件。这个资源文件通常是一个动态页面(如 ASP,PHP ,JSP 等文件)的网络地址(MVC 结构的程序例外)。Web 服务器根据动态页面文件的内容和 URL 中的参数,调用相应的资源(通过数据库或文件)组织数据,生成 HTML 页面。(注意这里生成的是一个 HTML 文档,里面可能包含 JavaScript 代码等,这里在服务器端不管 HTML 文档里的具体内容)

⑥ 生成 HTML 文档以后,服务器响应浏览器的 HTTP 请求,将生成的 HTML 文档发送给浏览器。

⑦ 浏览器接收请求得来的 HTML 文档。

⑧ 浏览器对 HTML 文档进行解悉,并请求 HTML 中链接的资源文件(如 JavaScript,CSS ,多媒体资源,内嵌网页)等。

⑨ 服务器接到浏览器对资源文件的 HTTP 请求以后,将相应的资源文件发送给浏览器。

⑩ 浏览器接收到请求来的资源文件,整理并呈现到页面中(浏览器将进行重排、重绘)。在进行页面呈现的时候,浏览器会从上到下执行 HTML 文档,当遇到相应的页面脚本的时候,会对脚本进行分析,并解释执行相应的脚本代码。当执行脚本时,将阻塞之后的链接文件的加载。

在此过程中, JavaScript 的加载会影响其他资源的加载。CSS 的加载虽然不会影响其他资源的加载,但是由于其与浏览器的渲染有关,因此加载会影响浏览器呈现和用户体验。资源加载方面,浏览器还会限制资源加载的并行连接数和最大连接数。

(2) HTTP 协议客户机与服务器信息交互

浏览器与服务器之间的 HTTP 通信协议对浏览器缓存机制进行了一系列规定,使得非首次访问不用重复加载加载过的资源,提高了资源的利用率,避免了重复加载的浪费。在 HTTP/1.0 中提供了一种简单的缓存机制,通过客户端根据获取文件中的日期和时间与在硬盘上缓存的文件进行对比。如果要获取的文件存在客户端缓存中,则客户端发送带有 If-Modifined-Since 字段的头部信息的 HTTP 请求。服务器程序若发现该文档没有发生改变,则返回 304 响应,不发生请求文件,此时客户端接收 304 响应后直接使用缓存中的文件。在这期间,浏览器就向服务器发送 HTTP 的请求及服务器接收到浏览器的发送的 HTTP 请求无可厚非是一个非常重要的过程。超文本传输协议 HTTP (HyperTextTransferProtocol) 是互联网上应用最为广泛的一种网络协议。设计 HTTP 最初的目的是为了提供一种发布和接收 HTML 页面的方法。通过 HTTP 或者 HTTPS 协议请求的资源由统一资源标识符来标识。HTTP 协议有四个版本: HTTP0.9、HTTP/1.0、HTTP/1.1 及 HTTPS,其中 HTTP/1.1 是当前最经常被使用的版本。它持久连接被默认采用,并能很好地配合代理服务器工作。还支持以管道方式在同时发送多个请求,以便降低线路负载,提高传输速度。HTTP/1.1 更改使用持续连接,不必为每个 Web 对象创建一个新的连接,一个连接可以传送多个对象。HTTP/1.1 还进行了带宽优化,例如 HTTP/1.1 引入 chunkedtransferencoding(分块传输编码)来允许 stream 化传输持续连接上发送的内容,取原先的 buffer 式传输[13] 。在 HTTP/1.0协议中每个 HTTP 请求都需要建立一个单独的连接,每次连接只传输单独的资源。在客户端和服务器每次建立和关闭连接中需要一定的时间消耗和延迟,频繁的请求影响客户端和服务器的性能。为了克服这个缺陷,HTTP/1.1 采用持久连接(也称作长连接),在同一个 TCP 连接上可以传送多个 HTTP 请求和响应,减少了建立和关闭连接的消耗和延迟。HTTP1.1 还允许客户端不用等待上一次请求结果返回,就可以发出下一次请求,但服务器端必须按照接收到客户端请求的先后顺序依次回送响应

结果,以保证客户端能够区分出每次请求的响应内容,这样也显著地减少了整个下载过程所需要的时间。

基于 HTTP1.1 协议的客户机与服务器的信息交换过程如图 1 所示。

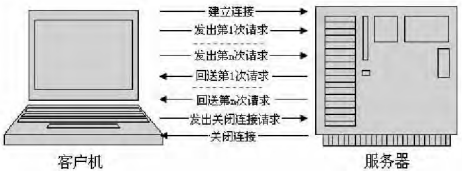

图 1　基于 HTTP1.1 客户机与服务器信息交互图

此外,HTTP 也有状态码和一些缓存的机制。HTTP 缓存可以简单地理解为: 当 Web 请求抵达缓存时,如果本地有"已缓存的"副本,就可以从本地存储设备而不是从原始服务器中提取这个文档。这样一来,就可以减少冗余的数据传输,减少了服务器的负担,大大提高了网站的性能,从而加快了客户端加载网页的速度。在 HTTP/1.0 中提供了一种简单的缓存机制,即客户端根据获取文件中的日期和时间与在硬盘上缓存的 HTTP 文件相对比。如果要获取的文件存在客户端缓存中,则客户端发送带有 If-Modifined-Since 字段的头部信息的 HTTP 请求。服务器程序若发现该文档没有发生改变,则返回 304 响应,不发生请求文件,此时客户端接收 304 响应后直接使用缓存中的文件[17]。HTTP/1.1 对缓存进行了加强,并使用了复杂的算法以及机制来实现缓存(过期验证模型和有效性确认模型),对与经常访问的客户端的请求响应产生的数据流量大大下降,降低服务器上拥塞的产生,同时大大缩短了响应时间,提高了服务器和 Web 前端的性能。

在服务器响应请求发送的过程中,浏览器与服务器可以进行 gzip 等约定的双方的压缩协议。

(3) 浏览器方面

在 B/S 结构结构中,Web 浏览器作为客户端最主要的应用软件,客户端统一化,将系统功能实现的核心部分集中到服务器上,简化了系统的开发、维护和使用。Web 浏览器可以对页面资源进行并行加载,但是对同一个域名下会有并行连接数的限制,同时对浏览器的总连接数也有所限制。不同的浏览器有着不同限制,图 2 是国外的社区驱动型项目 Browserscope 浏览器对比的结果[18]。

Top Browsers name	score	PerfTiming	Connections per Hostname	Max Connections
Chrome 24 →	12/16	yes	6	9
Firefox 18 →	13/16	yes	6	10
IE 8 →	7/16	no	6	35
IE 9 →	12/16	yes	6	35
IE 10 →	12/16	yes	8	16
Opera 12.11 →	10/16	no	6	16
Safari 6.0.2 →	11/16	no	6	9
Chrome 25 →	12/16	yes	6	9
Chrome 26 →	12/16	yes	6	9
Firefox 19 →	13/16	yes	6	16
Firefox 20 →	11/16	yes	6	12
Firefox 21 →	11/16	yes	6	9
Opera 12.12 →	10/16	no	6	9

图 2　常见浏览器的并行连接数与最大连接数

因此,应通过适当增加域名来提高同一页面的并行连接数、在域名解析、同一域下并行连接数和最大连接数之间适当的平衡。

在浏览器对服务器进行页面请求的过程中,JavaScript 的加载会影响其他资源的加载。CSS 的加载虽然不会影响其他资源的加载,但是由于其与浏览器的渲染有关,因此加载会影响浏览器呈现和用户体验。资源加载方面,浏览器还会限制资源加载的并行连接数和最大连接数。

1.2　服务器端优化

(1) 域名优化

浏览器限制并行连接数使得在同一个域名下加载的资源数量及其有限。对于页面请求很多的页面来说,页面加载速度必然受到限制。

采用多域名加载资源,可以提高同一时间内加载资源的数量,最大限度地利用有限的带宽。同时,采用多域名的另外一个好处在于,当 HTML 资源携带的 Cookie 过大时,将 HTML 规划到单独的域名,避免其他资源的请求来回传输很大的 cookie,从而避免了 HTTP 请求大小,提高 Web 前端性能。

局限性: 采用多域名不得不面临的问题是域名查找 DNS 的开销,DNS 解析可能需要几秒的时间,因此如何在域名解析时间消耗和多域名提高并行连接数折中,是采用多域名优化加载资源的考虑因素之一。

(2) 设置缓存

浏览器使用缓存来减少 HTTP 请求的大小,避免重复加载资源,提高资源的利用率、减少服务器压力、提高前端性能。在服务器与 Web 浏览器之间可以设置 Expires、Cache-Control(Max-Age)、mod_expires、Etag 头部信息等方式进行缓存。

设置 Expires 头部是最常用的方式之一,通过指定 HTTP 头部的 Expires 字段一个过期时间来验证是否使用缓存数据。其局限性在于,Expires 头部指定的时间,需要服务器与浏览器进行时间同步。

设置 Cache-Control 的 max-age 来进行缓存,可以避免与服务器进行实践同步。不过其只支持 HTTP1.1。通过同时设置这两个字段,可以在 HTTP1.1 时使用 Cache-Control,在其他使用 Expires 头部进行缓存。

此外,ETag 用于检测浏览器缓存中的组件是否与服务器上的组件一致。设置或移除 ETag 头部信息,可以有很大的作用,比如 Etag 对于 CacheCGI 页面很有用。特别是论坛,论坛有办法为每个帖子页面生成唯一的 Etag,在帖子未改变时,查看话题属性比较 Etag 就能避免刷新帖子,减少 CGI 操作和网络传输,减少论坛负担。适时移除 ETag 将有利于提升性能,避免数据存在于缓存时进行不必要的和低效的下载。但它也有一定的局限性: ETag 降低了代理缓存的效率,用户与代理之间的 ETag 经常不匹配。而通常 Last-Modified 与 ETag 是可以一起使用的,服务器会优先验证 ETag,一致的情况下,才会继续比对 Last-Modified,最后才决定是否返回 304。且在一般情况下,Cache-Control/Expires 会配合 Last-Modified/ETag 一起使用,因为即使服务器设置缓存时间,当用户点击"刷新"按钮时,浏览器会忽略缓存继续向服务器发送请求,这时 Last-Modified/ETag 将能够很好利用 304,从而减少响应开销。

(3) 使用 CDN

静态内容(比如图片、CSS、JavaScript、以及其他 cookie 无关的网页内容)都应该放在一个不需要使用 cookie 的独立域名之

上。因为域名之下如果有 cookie，那么客户端向该域名发出的每次 HTTP 请求，都会附上 cookie 内容。这里的一个好方法就是使用“内容分发网络”(ContentDeliveryNetwork，CDN)。

CDN 系统能够实时地根据网络流量和各节点的连接、负载状况以及到用户的距离和响应时间等综合信息将用户的请求重新导向离用户最近的服务节点上。CDN 正是出于缩短资源服务器与用户之间的距离而出现的，提高了资源的加载速度。

(4) 启动 Gzip 压缩

在发送文本内容之前可以对内容进行压缩，浏览器支持 Gzip、Deflate 等压缩技术，而 HTTP 协议上的 GZIP 编码是一种用来改进 Web 应用程序性能的技术。大流量的 Web 站点常常使用 GZIP 压缩技术来让用户感受更快的速度。Gzip 开启以后会将输出到用户浏览器的数据进行压缩处理，因此通过在服务器压缩再发送给浏览器，可以减少文本资源的大小，提高响应速度和加载时间。表 1 为 YSLow 对 http://join.qq.com 测试有无 Gzip 的结果。

表 1 使用 YSLow 对 http://join.qq.com 的测试有无 Gzip 结果

TYPE	SIZE(KB)	GZIP(KB)
doc(1)	6.0	
doc	6.0	2.0
js(6)	131.1	
js	3.7	1.2
js	11.4	4.0
js	30.8	15.5
js	18.4	4.9
js	22.0	7.1
js	44.4	13.6
CSS(2)	33.0	
CSS	17.1	4.1
CSS	15.8	3.4

局限性：一般对于 HTML、CSS、JavaScript 等文本内容为主的文件，Gzip 算法的效率比较高，而图片、pdf 等二进制文件使用 Gzip 的成效就不明显。而且 Gzip 压缩需要消耗服务器的资源，而解压缩需要消耗浏览器的资源，对于比较大的二进制文件具有非常大的性能消耗。过小的文件(通常小于 150 个字节)不宜进行 Gzip 压缩。

1.3 HTML 优化

(1) 减小 HTML 的大小

HTML 中存在着不少的空格字符，如果能去除这些空格字符，就可以减小 HTML 文件大小。HTML 文件缩小的工具有 HTMLTidy 或者自己用动态语言去除多余空格、制表符、换行、注释，以达到压缩 HTML 文件的减少文件的大小，直接提高了网页的加载速度。减少 HTML 大小，可以通过代码级压缩，去除空格字符或注释等无关的字符。精简代码，减少 HTML 元素的数量，注重 HTML 语义化，避免嵌套多层，在减少 HTML 文件大小的同时，还提高了页面渲染速度，提高了网页性能。

(2) 检查不存在或链接不到的资源

HTTP 请求、响应都需要耗费浏览器、服务器的性能，发出不带任何益处的请求降低了网页的性能。如果请求一个不存在的 JavaScript 文件，并且请求的服务器返回一个错误页面而不是 JavaScript 文件，那么浏览器会等待错误页面的内容下载完毕后才继续对所在页面进行解析，这样的空链接使得情况更糟。

(3) 减小 Cookie 的大小

浏览器对同域下的每个 HTTP 请求都会发送与该域有关的 Cookie 数据，因此在 Cookie 比较大时 HTML 页面采用独立域名有利于提高 Web 性能。

表 2 为 HTML 页面 Cookie 对响应时间的影响的实验结果。从该表中可以看出：在慢网速下，一个 3000 byte 的 Cookie 或者所有 Cookie 的容量为 3000 bytes 将会增加近 1 s 的延迟等待时间。这个结果进一步说明我们应该尽最大的可能减少 Cookie 的大小，从而较少其对页面响应时间的影响。

表 2 不同的 Cookie 大小对响应时间的影响

Cookie Size	Waiting	Receiving
0 B	679 ms	140 ms
500 B	876 ms	143 ms
1000 B	1.13 s	138 ms
1500 B	1.28 s	141 ms
2000 B	1.43 s	141 ms
2500 B	1.44 s	141 ms
3000 B	1.73 s	144 ms

1.4 JavaScript 优化

(1) 压缩 JavaScript 文件大小

与压缩 HTML 文件类似，JavaScript 通过去除所有的注释以及不必要的空白符达到精简，缩小 JavaScript 文件大小的目的。除此之外，JavaScript 还可以通过混淆，改写代码，将函数和变量的名字转换成更短的字符串，增加了反向工程的难度，同时大大减小了代码的大小。精简和混淆可以使用 JSmin 工具。

(2) 将 JavaScript 置于底部

JavaScript 脚本会阻塞浏览器的并行下载(某些高级版本的浏览器会有所优化，不会造成该现象)，阻塞后续组件的下载和对其后面内容的呈现。将脚本放在底部可以避免阻塞对页面的呈现和避免阻塞资源加载，提高网页响应速度和加载速度，并且可以让代码更加的干净，有利于蜘蛛的抓取。

(3) 采用无阻塞 JavaScript

使用 Ajax 加载脚本资源，加载完成后可以 eval 或者创建一个 script 的 DOM 元素，然后把响应的资源加入 script 元素内，然后嵌入 head。

通过 JavaScript 动态创建 script 的 DOM 元素，设置其 src 并附加到 HTML DOM 树上。该方法优于上述方法，该方法可以跨域获取脚本。

(4) 最小化重排和重绘

最小化重排和重绘可以通过合并 dom 操作和样式的修改：

a：通过 cssText 或 className 一次性修改多个样式。

b：通过设置 CSS 属性 display，隐藏元素，应用修改，然后重新显示

c：使用代码片段(documentFragment)在当前 DOM 之外构建一个 DOM 子树，DOM 操作完毕后，将其带回文档中。

d：将元素用于 DOM 操作的元素拷贝到一个脱离文档的节点中，DOM 操作完毕后，将其带回文档中。

(5) 取消依赖库和按需加载

在开发项目当中，有时为了提高开发效率，引入了很多 JavaScript 类库，而项目中只使用了其中非常少的一部分。这造

成过大的 JavaScript 下载,形成不必要的资源浪费。可以考虑使用精小的 JavaScript 库代替或者使用原生的 JavaScript 代替。

JavaScript 的加载会阻塞浏览器 UI 线程。如果将还未用到的 JavaScript 封装起来,但需要用到的时候再进行调用加载。或者进行预加载,有利于提高网页性能。

1.5　CSS 优化

(1) 精简 CSS

解析 CSS 的过程中会忽略文件中空格、制表符、换行符、格式化、注释等字符。除此之外,对部分 CSS(比如 font、background、margin 等)。同时在项目中经常会遇到写完 CSS 后没用到,造成不必要的 CSS 代码,这时可以通过 Chrome 的 Collect CSS Selector Profile 进行检查,并在代码中去除。

(2) CSS 位置

CSS 应该避免行内样式,采用外部样式。外部样式有利于维护、组件重用以及缓存,同时缩减 HTML 文件大小,在不采用样式的情况下,加载速度更快。

需将样式表放在 HEAD 标签中。浏览器是根据 CSS 结合 DOM Tree 进行渲染的。CSS 的加载影响网页渲染,置于 HEAD 开头有利于浏览器在解析 BODY 内容时,加载 CSS 以便达到逐渐呈现的效果,提高用户体验。

(3) 避免耗费性能的 CSS 写法

在 IE 中使用 CSS 表达式,它可能会被执行成千上万次,导致页面变慢。

在 CSS 的规范写法中,不同的选择器的解析性能是不一样的。样式系统从最右边的选择符开始向左边匹配规则。只要当前选择符的左边还有其他选择符,样式系统就会继续向左移动,直到找到和规则匹配的元素,或者因为不匹配而退出。根据这个匹配的规则。应该避免使用通配符、相邻兄弟选择符、子选择符、后代选择符和属性选择符等低效的选择区,更多考虑使用 ID 选择符、类选择符,规则越具体越好。

1.6　图片优化

(1) 减少小图片

在导航栏或按钮中,常常关联多个带有超链接的图片,这时可以考虑使用 ImageMap,而 CSSSprites 将一个页面涉及到的所有零星图片都包含到一张大图中去,当访问该页面时,载入的图片就不会像以前那样一幅一幅地慢慢显示出来了。对于当前网络流行的速度而言,不高于 200 KB 的单张图片的所需载入时间基本是差不多的,所以无需顾忌这个问题。利用 CSSSprites 能很好地减少了网页的 HTTP 请求,从而大大的提高了页面的性能。而且 CSSSprites 能够使图片之间更加紧凑,减少图片字节。所以,对于小图片,可以通过 ImageMap、CSSSprites、使用内联图片等方式合并多张小图片,来减少 HTTP 请求,减少网页加载时间,从而提高网页前端性能。

(2) 避免加载过大的图片

在 HTML 中,如果需要加载的图片尺寸过大或者过小,都可以通过前端 HTML 对其进行缩放。如果图片过大,便造成了不必要的图片下载开销。其次,可以选择提前将图片进行压缩到同比例的大小(可以通过图片工具或者在服务器端利用后台代码),从而选用适当比例的图片,有利于节省流量、减少下载开销。另外,选用适当的图片格式有利于提高网页质量和前端性能。

2　Web 性能优化实践

2.1　指尖点餐系统

指尖点餐系统是信息化移动生活中,将智能平板联合 HTML5 技术的发展用到点餐流程中,节约了餐厅的人力,提高餐厅的工作效率、经营收益和顾客满意度。该点餐项目基于 Web 新技术 HTML5 搭建,取代本地应用,缩短开发周期,更新周期短,维护成本低,硬件配置低等优点。

但其面临性能优化问题,在网速较慢的情况下,网页加载速度很慢,极大制约了该系统的使用和降低了顾客的满意度,这也是普遍 HTML5 应用相对于本地应用的弱势之一。

2.2　性能检测工具及测试环境

1) 网络层面上性能检测工具

(1) 采用各个浏览器的开发工具(如 firebug)查看代码、元素、加载情况以及清除缓存等。

(2) 利用 YSLow、pagespeed 等工具查看网络层面上的性能缺陷。

(3) 利用 Webkit 浏览器内置开发工具查看网络 network、Timeline。

(4) 利用 Fidder2 来限制网速,以及查看 HTTP 请求信息、请求汇总报告。

2) 浏览器层面上性能检测工具

(1) 利用 Webkit 浏览器内置开发工具的 profiles 监测 cpu、页面渲染、JavaScript 执行、CSS 利用率等。

(2) 利用 Speed Tracer 浏览器性能监测工具来监测浏览器 UI 线程、渲染重排、布局板面以及 JavaScript 执行效率代码进行级别优化等。

(3) 使用 Spirent 测试工具,采用平均事务响应时间、待测系统资源利用率和每秒事务数作为主要测试指标,综合评测 Web 应用的软硬件性能,并快速定位 Web 应用系统的性能瓶颈。

3) 测试环境

(1) 清空浏览器缓存。

(2) 清空 DNS 缓存(ipconfig /flushdns)。

(3) 慢网速下测试(使用 Fiddler 工具设置 512 KB 带宽),10M 网速下测试 UI Thread。

(4) 关闭不相关网页,开启监测工具(Speed Tracer)。

2.3　性能评测系统设计与实现。

(1) Web 应用评测过程

在 Web 应用评测中,每进行一轮测试时,需针对测试目标设计测试场景,通过 Spirent 实施一次负载测试,统计分析各项指标,生成测试报告,根据需要分析查找系统性能瓶颈,对相关瓶颈进行优化,如图 3 所示。根据需要和代价选择测试、优化的迭代次数,选择符合用户需求的最佳迭代方案。

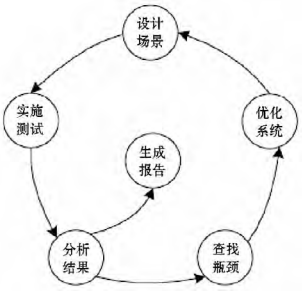

图 3　Web 应用评测过程

(2) 测试场景设计和选择

性能测试应该是一个全面的测试,测试目标系统各项能力、

发现系统的问题、定位系统瓶颈。如何设计测试用例，实现以最少的测试用例，最大发现系统的性能瓶颈，为优化提供指导。因此，既要从事务中的选取具有业务代表性的事件进行单独测试，又要对综合业务进行测试，防止因综合业务间事务或数据的相关性发生读写锁等待，并且设计测试用例时应尽量减少用例间的重叠，同时又无遗漏。

在指尖点餐系统中，最复杂的页面为点餐页面，该页面使用最频繁，结构最复杂，页面操作多，UI 交互多等特点。因此对该页面进行优化实践，对该系统最具现实意义。图 4 为该点餐系统的点餐页面截图。

图 4 自助点餐点餐界面

(3) 负载测试流程设计

Spirent 负载测试主要流程如图 5 所示，其核心是如何配置参数使测试能完成测试目标和对测试日志的分析，找出系统的瓶颈；负载由 Spirent 工具自动生成。

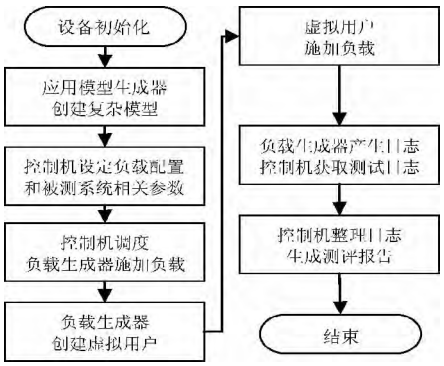

图 5 Spirent 负载测试的主要流程

2.4 寻找性能优化瓶颈

采用 HTML5 点餐应用项目中的点餐页面进行的测试，图 6 为点餐页面优化前首次加载的 Timeline。

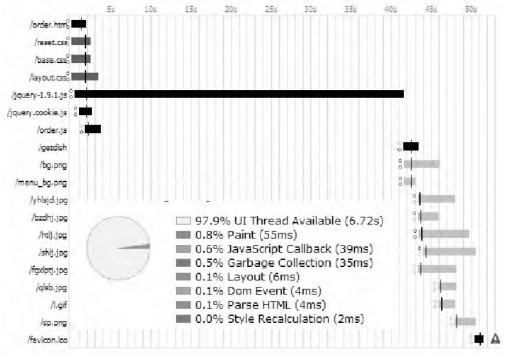

图 6 点餐页面优化前首次加载的 Timeline 和 UI Thread

图 6 为点餐页面采用 Fiddler 和 SpeedTracer 所检测到的 Timeline 和 UI Thread 拼合的测试数据。从该图中可以看出的问题如表 3 所示。

表 3 点餐页面优化前首次加载的 Timeline 结果分析

现象	分析
总体页面加载时间超过 50 秒	512 KB 带宽下(2G)的加载时间过长影响推广使用
jquery 单脚本文件加载超过 40 秒	资源过大，脚本阻塞加载
页面空白时间长达 41 秒	脚本加载执行导致页面渲染之后
资源并行加载上限 6 个	资源多，并行加载数量少
脚本文件差距较大，最大差距加载时间达 36 秒	脚本文件不合理
脚本执行为 39 ms 占据 UI Thread 比例 10%，10 M 网速下占据 19%	在网速较好、非首次加载时，优化提高、提高响应速度具有重大意义
页面渲染时间占据 UI 现存其余操作综合的 1 ~ 2 倍	页面渲染时间过长，DOM 树的重排重构过于频繁
1 个空请求，favicon. ico 图片找不到	浪费性能，请求等待的时间更长

所以针对页面渲染采取一定的性能优化措施尤为重要。

2.5 性能优化

(1) 服务器端优化

设置缓存后的二次加载后的二次加载如图 7 所示。

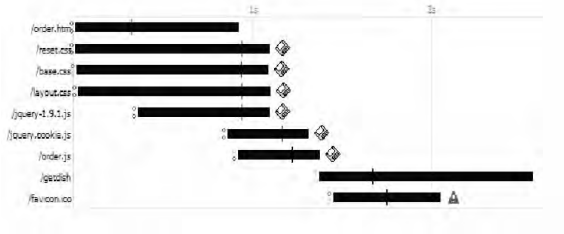

图 7 设置缓存期后第二次加载的 Timeline

开启 Gzip 压缩后的首次加载 Timeline 如图 8 所示。

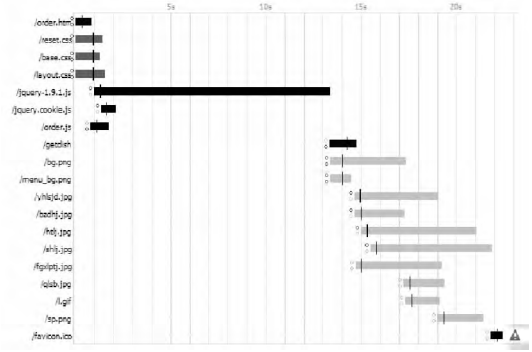

图 8 设置 Gzip 后首次的 Timeline

开启 Gzip 后加载时间较少了约 54%，总资源大小减少了约 42%，HTML/JS/CSS 等文件减少了 70% 左右。

(2) JavaScript 优化

针对该项目进行的 JavaScript 优化，主要精简代码，去除 jQuery 依赖。使得 JavaScript 文件大小从 82.9 KB 变到 3.8 KB。加载时间从 22 s 提高到 11 s，减少了 50% 的加载时间；脚本的

执行时间从 296 ms 变为 132 ms,减少了 54.5%。

同时通过优化 DOM 操作使页面渲染 Paint 减少了 34%(19 ms),DOM 事件响应时间减少了 71.5%(10 ms)。相应 Parse HTML 增加了 12 ms,这是由于采用 innerHTML 添加元素需要解析成 DOM 导致的。但是整体来说 UI 线程减少了 20 ms 左右。图 9 为进行 JavaScript 优化后的测试结果图。

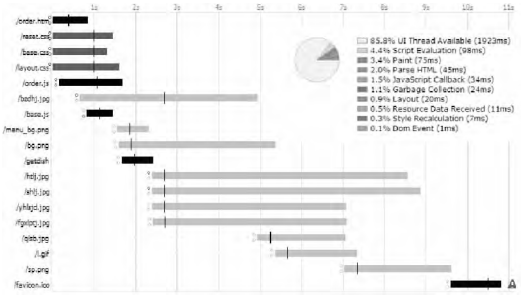

图 9　JavaScript 调整后的 Timeline 和 UI Thread

(3) JavaScript、HTML、CSS 组件优化

从项目之前的 Timeline 可以看出 CSS 的三个文件大小都比较小,分别为 1、1.4、2.6 KB。因此考虑将三个 CSS 小文件合并,以减少 HTTP 请求,提高加载资源速度。合并后的大小为 3.9 KB。文件大小减少了 1.1 KB,同时减少了 2 个 HTTP 请求,页面加载时间减少了约 1 秒。

对 JavaScript、HTML、CSS 组件进行代码级别压缩,去除可以去掉的字符以及对 JavaScript 进行混淆。页面大小减少了 1.8 KB 左右,页面加载时间缩短了约 0.5 秒,图 10 为操作后的效果图。

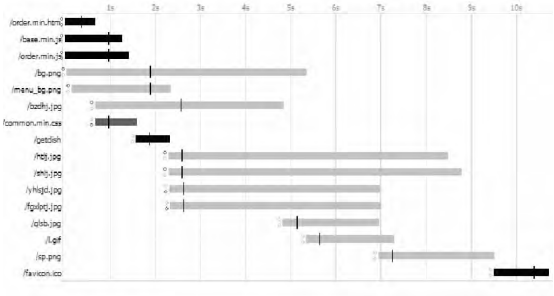

图 10　页面资源组件压缩后的 Timeline

(4) 图片优化

图片优化采取无损压缩,无损压缩后总体图片的大小减少了 8.8 KB,缩减了 5%。各个图片的无损压缩效果不一,比如 sp.png 压缩缩减了 26%,其余均为 3% 左右。页面整体加载速度提升了 40 ms 左右。相对于不同图片以及图片很多时,进行无损压缩可能带来意想不到的效果,不过对于本网页来说效果一般。图 11 为图片进行无损压缩后页面的 Timeline。

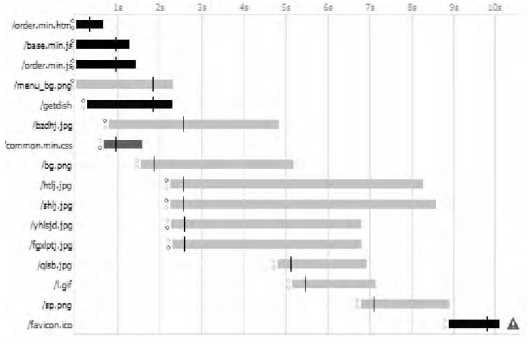

图 11　页面图片资源优化后的页面 Timeline

调整尺寸后,图片总量减少了 124.1 KB,缩减了 81%。网页加载速度提高了接近 2 s。如图 12 为优化图片尺寸后的 TimeLine,由此可见适当的时机对图片的尺寸进行优化,有利于提高网页性能,提高加载速度,避免加载过大资源造成浪费。

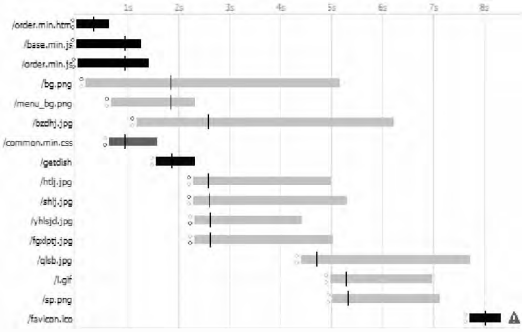

图 12　优化图片尺寸后的 Timeline

(5) 细节调整

从图 12 的 Timeline 中可以看出,页面中含有一个错误请求。该错误请求为 favicon。Favicon 可以让浏览器的收藏夹中、页面上方除显示相应的标题外,还以图标的方式区别不同的网站,很多网站并没有添加 favicon,因此造成了错误请求,浪费了资源。

2.6　优化前后对比

从表 4 点餐页面优化前后的对比数据分析中可以看出,加载时间的减少、页面渲染时间的减少极大提高了应用页面的呈现速度,页面脚本执行时间的降低提高了该应用的响应速度和整体性能。

表 4　点餐页面优化前后的对比的数据分析

	页面加载时间	页面渲染(Paint)时间	页面脚本执行时间
优化前	约 50 s	55 ms	39 ms
优化后	约 8.5 s	37 ms	12 ms
降低了	约 83%	约 32%	约 69%

3　结　语

本文研究了与 Web 前端相关的理论,归纳影响 Web 前端性能的因素,并对这些因素进行分析提出整体性、通用性的解决方案,具体如下:

(1) 首先对于 Web 前端相关的环境以及技术作为起点,研究与前端性能相关的机制或环境的原理,并分析其对 Web 前端性能的影响有哪些以及如何影响。

(2) 针对分析与 Web 前端性能有关的因素造成的影响,提出整体性(服务器方面、JavaScript、HTML、CSS、图片)的解决方案。归纳多种解决方案,并适当分析优劣、使用场合建议。有利于实际项目开展过程中针对不同方面提前进行优化。

(3) Web 前端性能不能仅仅局限于理论和测试,而且与业务密切相关。本文实现一个基于 HTML5 技术的 Web 移动应用,通过具体分析该项目的性能瓶颈,运用以上分析的理论和解决方案对其进行性能优化。使用性能以及网络检测工具检测通过具体解决方案进行优化前后的对比,并进行分析,从而提高了本文前端优化方案的可行性、真实性,丰富了 Web 性能解决方案的具体实施。

(下转第 147 页)

其中,α 称为原保险人的自留比例; V 是险种的协方差矩阵; l 为附加保费; $\alpha^{T}V\alpha$ 代表保险公司的总风险; c 代表原保险公司分保之后经过一个阶段期望达到的收益。通过调节参数 $\mu>0$ 的大小求解如式(1)的非线性方程组。

在熵一均值一方差优化模型中,用熵函数和方差共同作为风险的度量指标,也就是模型的目标函数,与均值一方差模型相比,这个模型具有明显的不同,把熵引进了风险度量领域中,使风险的度量更加合理化[8]。

3　相关设备性能容量评估

3.1　性能评估

每日数据接口批量性能: 数据接口主要包括从统一系统管理平台(UMP)接收个险保单数据。考虑公司未来三年业绩增长,估算个险保单数第三年每日增量约 4000 笔数据,数据量约为 40 MB,预估数据接收时间约在 10 分钟内,于次日上午 4 点前完成数据接收,符合业务要求。

每日个险再保险计算批量处理性能: 考虑公司未来三年业绩增长,估算第三年每日需再保计算保单约 4000 笔数据。由于不同业务种类的再保计算依赖其复杂度和处理方法,故具体计算时间存在差异。假设采用 2 CPU,4 GB 内存的数据库服务器,考虑再保计算中数据获取、计算和结果返回等全部处理环节,预估每个再保计算处理时间约为 0.5 秒,预估每日处理总时间为 $0.5\ s\times4000=2000\ s$,约 34 分钟,于次日上午 8 点前完成计算,符合业务需求。

3.2　容量评估

数据量年增长趋势: 公司现有存量需再保的个险保单约 10 万笔,考虑公司未来三年业绩增长,预估个险保单数第一年日增量约 1000 笔,第二年日增量约 2000 笔,第三年日增量约 4000 笔,三年增量约 185 万笔。参考现有个险保单再保比例,预估再保数据第一年日新增约 500 笔交易,第二年日增量约 1000 笔交易,第三年日增量约 2000 笔交易,三年增量约 92 万笔。

存储空间年增长趋势: 本系统需移行公司存量个险保单、存量再保险数据和产品信息,存量再保险保单约 10 万笔,存量再保数据约 20 万笔,总数据量约 3 GB; 增量数据约为:

1) 系统三年日均接收新增个险保单约 2350 笔,数据量约 23.5 MB;

2) 系统每日处理约 1170 笔再保数据,预估日增数据量约 11.7 MB。故每日数据增量约 35 MB 左右,年数据增量约 9.2 GB 左右。

本系统用户为公司精算用户、核保用户和系统管理员,目前总用户数约 20 人,年用户增量约 1 人。最大在线用户数 20 人(100% 使用),最大并发数 5 人。

4　结　语

经过测试,本设计已经实现了个险再保险信息管理系统的基本功能,系统平均响应时间不超过 3 秒。本设计的创新点在于直接面向保险行业,提出了再保险业务的网络化和信息化管理的一种技术解决方案,充分考虑到公司现有统一系统管理平台(UMP)、个人保险管理平台(PIMP)、财务核算管理平台(FAMP)的独立性和特殊性,通过公司内部平台技术框架,以 Java 应用基础为核心,搭建了一个完整的个险再保险信息管理系统,完成了个险再保险计算和业务管理等服务,形成了具有自主知识产权的个险再保险信息化产品。

参　考　文　献

[1] 丁慧莹. 医疗保险管理系统的研究与实现[D]. 吉林大学,2012.

[2] 胡炳志,陈之楚. 再保险[M]. 北京: 中国金融出版社,2006.

[3] 龙文,杨海珍,李晶,等. 从国际比较的视角看中国再保险市场发展前景[J]. 统计与决策,2010(13):108-111.

[4] 付晓宇. 银行保险信息管理系统的设计与实现[D]. 大连理工大学,2008.

[5] 杨依华. WebServices 技术在保险业务集成系统中的研究与应用[D]. 电子科技大学,2012.

[6] 王正宏,李小平. 基于 J2EE 架构的五层 Web 开发模型研究[J]. 现代商贸工业,2008(3):272-274.

[7] 何楠楠. 再保险的最优自留模型分析[D]. 山东大学,2010.

[8] 赵秀菊. 再保险的信息熵方法[D]. 大连理工大学,2006.

(上接第 95 页)

参　考　文　献

[1] Softpedia. The Average Web Page Loads in 2.45 Seconds Google Reveals [EB/OL]. http://news.softpedia.com/news/The-Average-Web-Page-Loads-in-2-45-Seconds-Google-Reveals-265446.shtml.

[2] Marketingland. Top Retail Websites Not Getting Faster: Average Web Page Load Time Is 7.25 Seconds [EB/OL]. http://marketingland.com/retail-website-load-times-continue-to-decline-with-a-22-decrease-during-the-last-year-37604.

[3] Onlinegraduateprograms. Instant America Network Search [EB/OL]. http://www.onlinegraduateprograms.com/instant-america/.

[4] KissMetrics. How Loading Time Affects Your Bottom Line [EB/OL]. http://blog.kissmetrics.com/loading-time/?wide=1.

[5] netcraft. April 2013 Web Server Survey [EB/OL]. http://news.netcraft.com/archives/2013/04/02/april-2013-web-server-survey.html.

[6] 李成银. 百度前端性能监控与优化实践 [EB/OL]. http://www.d2forum.org/.

[7] 操秀英,唐婷. 中国互联网为何"跑"不出世界的网速[N]. 科技日报,2010,10(3):1.

[8] 刘斌. HTML5-未来网络应用的核心技术研究[J]. 自动化与仪器仪表,2010(4):30-33.

[9] Steve Sounder. High Performance Web Sites[M]. 2007:1-170.

[10] Steve Sounder,et al. Even Faster Web Sites: Performance Best Practices for Web Developers[M]. O'Reilly Media,2009:1-250.

[11] 张紫微. Web 前端性能优化的研究与应用[D]. 成都: 电子科技大学,2010.

[12] 高克立. Web 性能的相关技术分析和研究[J]. 现代电信科技,2003(11):46-58.

[13] 周鹏,周海鹰,左德承,等. 基于 Spirent 的 Web 应用性能评测[J]. 计算机工程,2012,38(24):57-61.

[14] 聂应高. 基于 Page Speed 的网站前端性能优化分析[J]. 现代图书情报技术,2009(11):69-73.

[15] 连志刚. Web 系统性能测试过程模型研究[D]. 西安: 西北大学,2012.

[16] 赵佳佳. Web 性能测试与瓶颈分析的研究[D]. 长春: 长春理工大学,2012.

[17] 高洁璇. Web 管理信息系统性能优化研究[D]. 武汉: 华中科技大学,2011.

[18] Browserscope [EB/OL]. http://www.browserscope.org/.