计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200600038

基于 Web 的数据可视化图表渲染优化方法

鄂海红　张田宇　宋美娜
北京邮电大学计算机学院　北京 100876
(ehaihong@bupt.edu.cn)

摘　要　在数据可视化场景中,Web 页面作为数据可视化图表的承载主体,其性能的好坏直接影响可视化图表的加载速度和渲染效果,而目前基于 Web 技术的优化手段不能很好地缓解图表获取大规模复杂数据并进行渲染重绘所造成的网络数据传输压力。针对上述问题,提出了一种基于 Web 的数据可视化图表渲染方法。首先,将缓存机制与增量更新算法相融合,并从图表样式及交互配置信息和图表绑定数据两个方面对 HTTP 请求响应体进行深度优化。然后,通过减小 HTTP 请求响应体的大小来降低网络数据传输量,缩短数据资源的下载时间,进而提升图表加载速度,缩短页面渲染时长。最后,针对该方法进行充分的对比实验,实验结果表明,单个图表 HTTP 的总响应时间由 75 ms 缩短至 28 ms,Web 页面展示多个图表的总渲染时长由 1546 ms 缩短至 1337 ms,从而验证了该方法的有效性。

关键词:Web 性能优化;数据可视化;图表渲染优化;HTTP 响应;Web Storage;增量更新
中图法分类号　TP302

Web-based Data Visualization Chart Rendering Optimization Method

E Hai-hong, ZHANG Tian-yu and SONG Mei-na
School of Computing, Beijing University of Posts and Telecommunications, Beijing 100876, China

Abstract　In data visualization scenario, as load-bearing body of data visualization, the web page's performance directly affects the loading speed and rendering effect of the visualization chart. At present, the optimization method based on web technology cannot reduce the pressure of network data transmission caused by charts obtaining large-scale complex data for rendering and redrawing. In view of the above problems, a web-based data visualization chart rendering method is proposed. Firstly, it combines the caching mechanism and the incremental update algorithm, and deeply optimizes the HTTP request response body from the aspects of chart style and its interactive configuration information and chart binding data. Then, by reducing the size of the HTTP request response body, it reduces the amount of network data transmission and shortens the download time of data resources by reducing the size of the HTTP request response body, thereby improving the chart loading speed and shortening the page rendering time. Finally, a full comparison experiment is carried out for this method. Experimental results show that the total HTTP response time of a single chart is shortened from 75 ms to 28 ms, and the total rendering time of multiple charts displayed on web pages is shortened from 1546 ms to 1337 ms, thus the effectiveness of this method is verified.

Keywords　Web performance optimization, Data visualization, Chart rendering optimization, HTTP response, Web Storage, Incremental update

1　引言

数据可视化作为大数据分析最热门的研究领域之一,能够利用类型丰富的图表等视觉元素,针对多来源复杂数据进行数据可视化展示,便于挖掘数据中隐含的信息并发现数据中所包含的规律[1],以实现对复杂数据的全面且深层次的探索和分析。目前数据可视化技术正逐步与数据挖掘及分析技术相结合,为数据分析奠定了坚实的基础[2]。

Web 页面作为数据可视化技术展现的主要载体[3-7],为了对数据进行全方位、深层次的探索挖掘和实时数据监控,不仅需要在页面中展示众多数据复杂且规模庞大的图表,还需要不断更新图表数据,以达到数据的实时呈现,因此 Web 性能的好坏对于可视化图表的加载速度和渲染效果尤为重要。

然而,目前针对 Web 的性能优化手段主要集中在减少并压缩 http 请求、改进网络协议以及利用负载均衡技术处理和解析并发请求等方面[8-12]。Song 等[13]提出一种基于 http 请求的调度优化方案,从而解决了浏览器二连接限制,缩短了 http 的总体请求回复时间。Xu 等[14]利用 http/2 协议的特性来解决连接复用率低的问题,并通过重复数据传输来优化 web 页面的加载时间。Chen 等[15]从减小网络通信量和优化

到稿日期:2020-06-04　返修日期:2020-08-15　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点研发计划(2018YFB1403501)
This work was supported by the National Key R&D Program of China (2018YFB1403501).
通信作者:张田宇(zhangtianyu0821@bupt.edu.cn)

数据访问等方面探讨了如何优化 Web 应用系统的性能。这些技术手段能够通过减小服务器端压力和缩短请求时间等方式来优化页面性能，但是数据可视化图表频繁获取大规模图表数据并进行渲染重绘所造成的网络数据传输压力却不能得到很好的缓解。

在网络传输数据进行图表可视化展示的过程中，如果要传输的数据量规模庞大，则极易造成数据资源下载时间过长，使得图表加载缓慢，从而导致页面渲染时间延长。因此，本文提出一种针对数据可视化图表的渲染优化方法，旨在通过减小 http 请求响应体中包含的数据量来减小网络中的数据传输规模，能够在一定程度上缩短响应数据资源的下载时间，进而提升图表的加载速度，缩短页面渲染时长。

2 数据可视化图表渲染优化方法

在数据可视化应用场景下，往往需要在一个 Web 页面同时展示几个甚至几十个不同类型的复杂图表，同时为了实现数据的实时监控，还需要频繁地更新每个图表的数据，随着图表数目的增多，数据规模会愈发增大，导致网络传输压力过大，进而影响图表的加载速度。

然而，页面在加载图表到渲染展示的过程中不仅需要获取大规模的复杂数据，同时为了实现图表的可交互性及美观，还需要大量的图表样式及交互动画的配置信息。以目前广泛使用的可视化工具 Echarts[16] 为例，一个图表通过 options 接口不仅传递了大量的图表样式及交互配置，同时还通过 series 属性绑定了图表要展示的数据。

本文提出的数据可视化图表的渲染优化方法，从图表样式及交互配置信息和图表绑定数据两方面来优化 http 请求响应体的数据传输，解决了可视化图表展示过程中遇到的性能问题。通过分析数据图表在实时更新过程中数据的变化发现，图表样式及交互配置信息并未随数据变化而频繁更新，并且图表每次绑定的数据更新改动较小，因此本文将数据可视化图表渲染优化方法分为两部分：1）利用缓存机制存储多个图表的样式及交互信息，图表每次更新时可直接从缓存中获取，无需占用 http 请求响应体空间；2）利用增量更新算法，计算图表更新前后数据差异生成的很小的补丁数据，用这些补丁数据来代替大规模的图表数据在网络中进行传输。数据可视化图表渲染优化方法的总体设计图如图 1 所示。

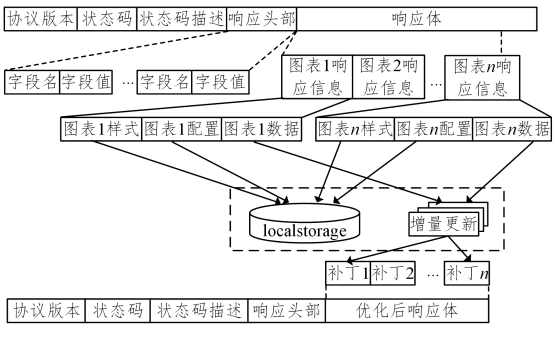


图 1 数据可视化图表渲染优化方法的总体设计图

Fig. 1 General design diagram of rendering optimization method for data visualization charts

2.1 基于缓存的多图表渲染优化

缓存作为 Web 页面优化的重要途径之一[17-18]，其实现方式主要分为两种：1）HTTP 协议中提供的缓存方案，通过设置 http 请求头部的相关字段来选择缓存策略，可以将 css 样式表、js 脚本、图片资源等静态资源缓存在浏览器端，以提高资源利用率和网页加载速度；2）浏览器本地缓存方案，常用的有 cookie，localStorage，sessionStorage 等，通过读取本地存储中的数据来避免与服务器的频繁通信，从而减少 http 请求发送的数目，提升 Web 页面的性能[19]。

由于 http 缓存方案在重复请求静态资源时依旧会向服务器发起 http 请求，询问缓存是否有效，当资源未更新时会造成不必要的网络开销，因此对于不需要频繁更新的资源，在一定程度上可利用浏览器缓存方案来替代。

本文考虑利用浏览器缓存中的 localStorage 对同一个 Web 页面下的多个图表样式及交互配置信息进行本地存储，从而减小网络中的数据传输量，缩短数据资源下载时间。基于缓存的多图表渲染优化方法的总流程如图 2 所示。

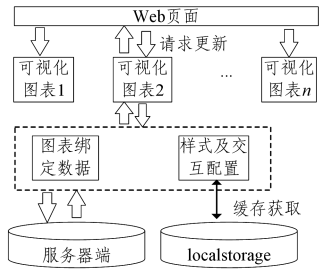


图 2 基于缓存的多图表更新流程

Fig. 2 Multi-chart update process based on local storage

当页面请求更新多个图表时，其中图表所绑定的数据需要通过发送 http 请求来获取每次更新后的数据响应，图表的样式及交互配置则通过读取缓存来获取，无需占用网络进行传输，这样 http 请求的响应体中只包含了图表所绑定的数据，一定程度上减小了响应体的数据规模，节省了网络带宽。

2.2 基于大规模图表数据的增量更新算法

在利用图表进行数据可视化展示的过程中，除了样式和交互等配置信息会占用大量网络资源外，图表绑定的数据规模也不容小觑，并且对于一些绑定了大量数据的复杂图表，在每次图表更新的过程中，都需要重新将数据提供方更新后的数据通过网络传输重新获取，然后再次进行图表的渲染绘制。然而，对于其中的一部分图表来说，更新后的数据相比原始数据只改变了某个字段下的几个数值，即便如此也需要重新下载整套数据，这种大规模数据的频繁传输势必会引起资源下载时间过长，从而导致图表加载非常缓慢，甚至出现卡顿。

针对上述问题，本文提出了一种增量更新算法，针对数据规模较大，并且更新前后数据改动较小的情况，通过计算新旧数据的差异，获得存储了数据修改记录的补丁数据，将该补丁数据通过网络传输给 Web 页面，然后利用补丁数据和原始数据进行计算，生成新数据，以替换页面中所展示图表的绑定数据。下面详细介绍本文提出的增量更新算法。

该算法主要分为两部分：

（1）利用最长公共子序列算法（Longest Common Subsequence）及其回溯方法求出记录新旧数据差异的补丁数据

LCS 算法[20-21] 是在两个给定序列中寻找以相同顺序出现的子序列，子序列在给定序列中的位置可以不连续。该算法利用动态优化的思想，将大问题分解为小问题来求得最优解。

定义 1 设 X 序列长度为 m，Y 序列长度为 n，C 为一个 $(m+1)*(n+1)$ 的矩阵，其中矩阵的第一行和第一列均置为 0，$C[i][j]$ 表示字符序列 X 和 Y 中前 i 和前 j 个字符的 LCS 长度，其计算式如式(1)所示：

$$C[i][j]=\begin{cases}0, & \text{if } i=0 \text{ or } j=0\\ C[i-1][j-1]+1, & \text{if } x_i=y_j\\ \max\{C[i][j-1],C[i-1][j]\}, & \text{if } x_i\neq y_j\end{cases} \tag{1}$$

根据式(1)，本文给出了最长公共子序列的求解算法，如算法 1 所示。

算法 1 getLCS(X,Y)

输入：序列 X，序列 Y

输出：最长公共子序列矩阵 C(m+1,n+1)

```
1. 初始化：C 矩阵首行和首列置为 0
2. for i ← 1 to length[X] do
3.    for j ← 1 to length[Y] do
4.       if X[i−1]=Y [j−1] then
5.          C[i][j] ← C[i−1] [j−1]+1
6.       else if C[i−1][j] >= C[i][j−1] then
7.          C[i][j] ← C[i−1][j]
8.       else
9.          C[i][j] ← C[i][j−1]
10.   end for
11. end for
```

定义 2 设一个二维数组 B，使用↖，←，↑ 3 个字符来表示矩阵各单元格对应值的来源。↖表示 X 和 Y 同时扫过同一个字符，即公共子序列中的字符；←表示 Y 扫过一个字符，将此字符标记为新增；↑表示 X 扫过一个字符，此字符标记为删除。由此，可以得到式(2)：

$$B[i][j]=\begin{cases}\nwarrow, & C[i][j]=C[i-1][j-1]+1\\ \leftarrow, & C[i][j]=C[i][j-1]\\ \uparrow, & C[i][j]=C[i-1][j]\end{cases} \tag{2}$$

在算法 1 求得的最长公共子序列矩阵 $\boldsymbol{C}$ 的基础上，根据式(2)进行路径回溯，求得序列 X 更新为 Y 的最短编辑距离，记录为补丁数据，其中 −1 代表删除操作，+代表新增操作，1 代表字符未变化。如果有多个连续操作，可以考虑将其合并，如算法 2 所示。

算法 2 MinEditPatch(X,Y,$\boldsymbol{C}$)

输入：序列 X，序列 Y，最长公共子序列矩阵 C

输出：补丁数组 Patch[]

```
1. 初始化：i ← length[X],j ← length[Y]
2. while i > 0 and j > 0 do
3.    if X[i−1]=Y [j−1] then
4.       ▷insert 1 into Patch [⋯]
5.       i ← i−1,j ← j−1
6.    else if C[i−1][j] > C[i][j−1] then
7.       ▷insert −1 into Patch [⋯]
8.       i ← i−1
9.    else
10.      ▷insert "+Y[i−1]" into Patch [⋯]
11.      j ← j−1
12. end while
13. 将 X 序列中剩余所有字符记为删除
14. 将 Y 序列中剩余所有字符插入 Patch 数组中
```

假设 X=ABCABBA，Y=CBABAC，按照算法 1 计算矩阵对应位置的值，则 $C[m][n]$ 即为最长公共子序列长度。根据式(2)计算得到的箭头指示，在矩阵 $\boldsymbol{C}$ 的基础上可以得到一条从右下角到左上角的回溯路线，如图 3 所示。求得的最长公共子序列为 CBBA，长度为 4。在回溯的过程中，根据箭头表示即可得到 X 到 Y 的最短编辑距离，根据算法 2 将其记录在补丁数组 P 中，则 P=[−2,1,−1,1,+A,2,+C]，重复操作越多，补丁数组的长度越小，网络传输就越快。

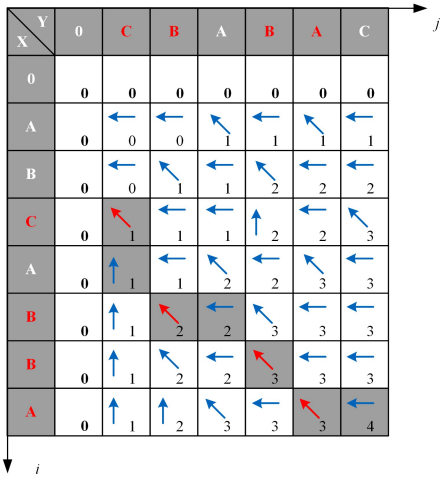


图 3 最小补丁数组计算示意图

Fig. 3 Calculation diagram of minimum patch array

(2)利用补丁数据修改原始数据得到更新后的数据

当用于展示图表的 Web 页面获取到服务器端由算法 1 和算法 2 联合计算得到的补丁数组后，在前端页面将图表原始数据与补丁数组合并生成新的数据，用于图表的更新重绘。该步骤的主要执行流程如图 4 所示。

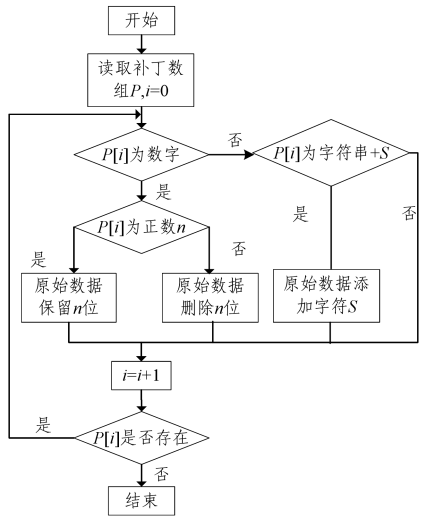


图 4 获取图表更新数据执行流程

Fig. 4 Execution flow for obtaining chart update data

根据图 4 所示的执行流程，本文提供补丁数据合并算法，并在原始序列的基础上求得更新后的序列，如算法 3 所示。

算法 3 PatchMerge(Old,Patch)

输入：原始序列 Old，补丁数组 Patch

输出：更新后序列 New

```
初始化：index←0
1. for i←0 to length[Patch] do
```

```
2.   if typeof Patch[i] is number then
3.     if Patch[i] > 0 then
4.       ▷insert Old [index,…,index+Patch[i]]into New […]
5.     end if
6.     index ← index+Patch[i]
7.   end if
8.   if typeof Patch[i] is string then
9.     ▷insert Patch[i] [1,…,length [Patch[i]] intoNew […]
10.  end if
11. end for
```

3 实验验证

本文实验使用 Chrome 浏览器的 TimeLine 工具查看页面加载事件和监测网络数据资源加载时间，同时使用 Chrome 的 Profiles 工具监测 CPU 的使用情况，以及页面渲染、脚本执行和图表绘制等时间消耗情况。

TimeLine 工具的主要检测参数如下：Waiting（Time to first byte (TTFB)），是发起最初的网络请求到服务器接收到第一个字节这段时间，其包含了 TCP 连接时间、发送 HTTP 请求时间和获得响应消息第一个字节的时间；Content Download，获取 Response 响应数据消耗的时间花费。

3.1 多图表渲染优化实验

根据 2.1 节提出的基于缓存的多图表渲染优化方法，将部分图表样式及交互配置信息存入 localStorage 缓存中，分别记录图表数目不同时资源等待时间和下载时间的变化趋势，实验结果如表 1 所列。为确保实验不受其他因素的影响，实验对象的图表均为首次加载渲染。

表 1 多图表数据传输性能的对比

Table 1 Transmission performance comparison of multi-chart data

Number of Charts	Method (original/optimized)	Waiting Time/ms	Download Time /ms
1	ORI	11.84	11.33
	OPT	11.84	0.65
5	ORI	12.42	25.82
	OPT	13.14	1.43
10	ORI	14.77	35.22
	OPT	12.60	1.19

从表 1 中的实验数据可以看出，资源等待时间没有明显变化，随着图表数目的增多，优化前的图表资源下载时间逐步增加，而优化后的图表资源下载时间基本维持在 1 ms 左右，没有较大波动，并且图表数目越多，优化前后的时间差异就越大。

3.2 单图表数据传输优化实验

根据 2.2 节提出的基于大规模图表数据的增量更新算法，图表每次发送更新请求时，服务器端接收请求并进行相应计算，在得到补丁数组后将其传回客户端，然后客户端根据补丁数组更新图表的原始数据。本文一共做了 5 组实验，实验结果如表 2 所列。从第一组实验数据可以看出，采用本文算法进行优化后，由于算法的计算耗时较长，使得数据资源等待时间相比优化前有所增加，但增幅很小，而资源下载时间从 58.11 ms 降到 0.56 ms，相比优化前有了大幅下降，最终导致优化后的总耗时远远低于优化前。

为避免实验结果的偶然性，在第一组实验的基础上增加了 4 组对比实验：第一组和第二组对比，更换了数据资源，资源下载时间由 44.71 ms 降到 0.99 ms，仍然有明显的优化结果。第三组－第五组在第二组数据的基础上，不断加大对图表原始数据的改动来进行实验对比，通过图 5 可以直观地看出，随着数据改动量的加大，采用增量更新方法优化后的实验结果依旧有一定的改善，但是优化前与优化后的总耗时差异在逐渐减小，并且优化后的等待时间和资源下载时间随着改动的加大而增加。

表 2 单图表数据传输性能对比

Table 2 Transmission performance comparison of single chart data

Experi-mental group	Method(original/optimized)	Waiting Time/ms	Download Time /ms	Total Time/ms
1	ORI	15.10	58.11	75
	OPT	22.22	**0.56**	28
2	ORI	10.77	44.71	56
	OPT	37.87	**0.99**	43
3	ORI	10.30	47.31	58
	OPT	38.29	**1.21**	50
4	ORI	10.56	48.79	60
	OPT	47.10	**2.91**	53
5	ORI	13.37	57.50	72
	OPT	56.20	**5.43**	71

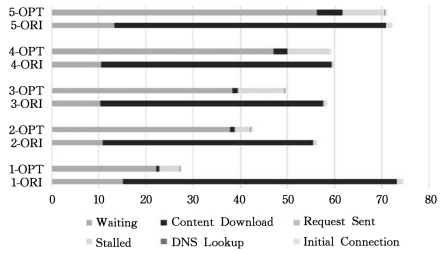


图 5 单图表数据传输性能的变化趋势

Fig. 5 Change trend of single chart data transmission performance

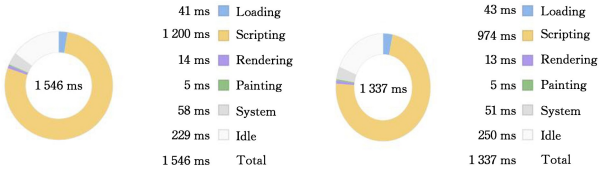
由于数据的变动越来越大，服务器端的计算更加耗时，使得页面资源等待时间过长，并且返回的补丁文件的大小也会随之增大，导致资源下载时间越来越长，则优化后的方案总耗时就会逐步接近于优化前的，在传输补丁数据的网络耗时与服务器端算法执行时间之和超过传输更新后整个图表数据的网络耗时的情况下，本文提出的优化方法不会取得明显的提升效果。

因此，本文基于大规模图表数据的增量更新算法，在数据改动规模较小的情况下，通过计算生成的补丁数据大小满足补丁数据资源下载时间与增量更新算法执行时间之和小于图表更新数据资源下载时间，说明图表数据传输性能有明显的提升；而对于大量数据的更新改动，生成的补丁数据的大小满足补丁数据资源下载时间与增量更新算法执行时间之和大于图表更新数据资源下载时间，那么使用增量更新算法计算的代价已经超过了对资源下载时间的优化，因此增量更新并不适用于大量数据的改动。

3.3 综合实验

将第 2 节中提出的基于缓存的多图表渲染优化方法与基于大规模图表数据的增量更新算法相融合，对 http 请求响应体进行深度优化，在一个 Web 页面展示相同图表的情况下（以展示 5 个图表为例），采用本文方法优化前后的页面渲染时间对比图如图 6 所示。通过对比可以看出，优化后的页面的总渲染时间由 1546 ms 降至 1337 ms，其中页面渲染的时间主要消耗在脚本计算上，将大量的样式和交互配置信息存入缓存，并通过增量更新来减小响应数据的大小，能够有效地降

低这一部分的时间，从而提升整个页面的加载速度。



(a)优化前 (b)优化后

图 6 页面渲染检测对比

Fig. 6 Page rendering detection comparison

结束语 本文提出一种基于 Web 的数据可视化图表渲染优化方法，从图表样式及交互配置信息和图表绑定数据两方面，分别利用浏览器缓存和增量更新算法，使得在每次图表实时更新展示的过程中，从缓存中获取图表样式及交互配置，从服务器端获取图表绑定数据更新前后的补丁数据，并根据补丁数据和原始数据还原更新后的数据，目的是减少网络中的数据传输量和节省网络带宽，从而缩短数据资源下载时间，提升图表的加载速度。本文实验部分针对提出的两种方法进行了充分的对比实验，实验结果表明，采用本文方法优化后的图表，从发送请求到图表渲染成功的总耗时明显缩短，证明了本文方法的有效性。

然而，本文方法也存在一定的不足，在更新前后图表绑定数据差异较大的情况下，由于算法计算耗时较长，使得优化前后的总耗时随着补丁数据的增大而越来越接近。因此，未来可考虑采用更加高效的增量更新算法，从而对大数据量的更新仍然有较好的提升效果。

参 考 文 献

[1] BIKAKIS N. Big Data Visualization Tools[C/OL]// Encyclopedia of Big Data Technologies. Springer, Cham. https://doi.org/10.1007/978-3-319-77525-8_109.

[2] REN Y G, YU G. Researeh and Development of the Data Visualization Techniques[J]. Computer Science, 2004(12): 92-96.

[3] LI Y, MA J M, AN B, et al. Web Based Lightweight Tool for Big Data Procesing and Visualization[J]. Computer Science, 2018, 45(9): 60-64, 93.

[4] DOLATABADI A D, NOURI H. Optimized Web Based Method for 2D-Visualization of 3D Medical Images[C]// World Congress on Engineering. 2014: 551-556.

[5] SECHLER J, HARRISON L, PECK E M, et al. SightLine: Building on the Web's Visualization Ecosystem[C]// Human Factors in Computing Systems. 2017: 2049-2055.

[6] BAE P, LIM K, JUNG W, et al. Practical implementation of M4 for web visualization service[J]. Journal of Communications and Networks, 2017, 19(4): 384-391.

[7] REBELO J, ANDRADE C, COSTA C, et al. An immersive web visualization platform for a big data context in bosch's industry 4.0 movement[C]// Information Systems-16th European, Mediterranean, and Middle Eastern Conference. 2020: 71-84.

[8] CAO B, SHI M, LI C, et al. The solution of web font-end performance optimization[C]// International Congress on Image and Signal Processing. 2017: 1-5.

[9] SHUANG K, ZHANG T, DONG Z, et al. Impact of HTTP Pipelining Mechanism for Web Browsing Optimization[C]// IEEE International Conference on Mobile Services. 2015: 415-422.

[10] SAKAMOTO Y, MATSUMOTO S, TOKUNAGA S, et al. Empirical study on effects of script minification and HTTP compression for traffic reduction[C]// International Conference on Digital Information, Networking, and Wireless Communications. 2015: 127-132.

[11] WOLSING K, RUTH J, WEHRLE K, et al. A performance perspective on web optimized protocol stacks: TCP+TLS+HTTP/2 vs. QUIC[J]. arXiv: Networking and Internet Architecture, 2019: 1-7.

[12] LI M Z, CHEN J, WANG J L, et al. HTTP Chunked Stream Concurrence Analysis Based on State Machine [J]. Computer Science, 2018, 45(9): 60-64, 93.

[13] SONG M N, WANG N, E H H. Research on Optimization Algorithms for HTTP Maximum Concurrent Connection Restriction [C]// 2018 International Conference on Information Systems and Computer Aided Education (ICISCAE). 2018.

[14] XU Z W, WANG Y. Research and Experiment on Relationship between New Features of HTTP/2 and Web Performance [J]. Computer Technology and Development, 2017, 27(11): 192-195.

[15] CHEN L B. Research on Performance Optimization Method of Web Application System Based on J2EE [J]. Computer Science, 2006(7): 95-97.

[16] LI D Q, MEI H H, SHEN Y, et al. ECharts: A declarative framework for rapid construction of web-based visualization[J]. Visual Informatics, 2018, 2(2): 136-146.

[17] ILIEV I, DIMITROV G P. Front end optimization methods and their effect[C]// 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics(MIPRO). IEEE, 2014: 467-473.

[18] WANG P P, WEI Z Q, LI Z. Front-end Website Performance Optimization Based on Web Storage in Html5[C]// Software Engineering and Information Technology. 2016: 36-40.

[19] SHEN D. Research on the Optimization Technology of the Font-End Performance of Web System Based on Bootstrap [D]. Beijing: Beijing University of Posts and Telecommunications, 2016.

[20] COSTAS S, ILIOPOULOS M, SOHEL R. Algorithms for Computing Variants of the Longest Common Subsequence Problem [J]. Theoretical Computer Science, 2008, 395(2): 255-267.

[21] ZHENG Z J, WANG H, YU C. Two Algorithms to Solve Longest Circular Common Subsequence P-roblems [J/OL]. Application Research of Computers: 1-5. [2020-05-23]. https://doi.org/10.19734/j.issn.1001-3695.2019.06.0258.

E Hai-hong, born in 1982, Ph.D, associate professor, is a member of China Computer Federation. Her main research interests include big data platform, cloud computing and microservice architecture.

ZHANG Tian-yu, born in 1996, postgraduate. Her main research interests include big data platform and data visualization.