УДК 004.414.22
DOI: 10.15587/2313-8416.2015.46992

МЕТОД ОПТИМІЗАЦІЇ ВІДОБРАЖЕННЯ ГРАФІЧНОЇ ІНФОРМАЦІЇ В КЛІЄНТ-СЕРВЕРНИХ ІНФОРМАЦІЙНИХ СИСТЕМАХ

© А. О. Болдак, Ю. В. Мазуревич

В роботі запропоновано метод скорочення часу завантаження та об'єму даних, необхідних для відображення веб сторінки за рахунок попередньої обробки на стороні сервера. Проведено оцінку ефективності реалізації такого способу. Виявлені умови, в яких він буде найбільш ефективний, а також його недоліки та наведено шляхи зменшення їх впливу.
Ключові слова: *оптимізація JavaScript, пришвидшення завантаження, інтерактивні графіки, попередня обробка, автоматична обробка коду*

This paper presents an approach to reduce load time and volume of data necessary to display web page due to server side preprocessing. Measurement of this approach's effectivity has been conducted. There were discovered conditions in which this approach will be the most effective, its disadvantages and presented ways to reduce them
Keywords: *JavaScript optimization, loading speedup, interactive plots, preprocessing, automatic code processing*

1. Вступ

Представлення даних у вигляді графіка – дуже хороший спосіб донесення даних до кінцевого користувача. Також цей спосіб полегшує знаходження закономірностей в даних, завдяки чому часто використовується не тільки для презентацій, а й як інструмент досліджень. Для використання графіків в дослідженнях важливу роль починає відігравати інтерактивність.

2. Аналіз можливостей для включення графіків в документ та постановка проблеми

Існує два способи генерації графічного відображення: на стороні клієнта та на стороні сервера.

Для реалізації побудови графіків на стороні сервера можуть використовуватись бібліотеки як:

JFreeChart – відкрита бібліотека для мови програмування Java, що спрощує створення різноманітних складних діаграм. Через різноманітні методи набору класів надає майже повний контроль над областю діаграми. Так реалізовані механізми збільшення/зменшення, обробки подій, можливості створення кількох діаграм на одній області, текстові підказки, задання вигляду кривих, точок та фону і т. п. [1].

Charts4j – це безкоштовний легковаговий Java API для побудови діаграм та графіків. Він дозволяє розробникам програмно створювати діаграми, що доступні в Google Chart Tools через простий та інтуїтивний Java API [2].

Такі бібліотеки орієнтовані на побудову статичних зображень, а ті, які підтримують інтерактивність, створені для використання в окремих додатках.

Таким чином, генерація графіків на стороні сервера – це створення растрових чи векторних зображень, що позбавляє відображення інтерактивності. А динамічна зміна можлива лише через нові запити до сервера.

Для вирішення цієї задачі на стороні клієнта можуть використовуватись:

Flot – це бібліотека для побудови графіків на чистому JavaScript для jQuery, з фокусом на простоту використання, привабливий вигляд та інтерактивні елементи;

Ember Charts – бібліотека для створення графіків, що побудована на Ember.js та d3.js фреймворках. Вона включає часові, стовпчикові, кругові та точкові діаграми, які легко використовувати та модифікувати [3].

Використання JavaScript бібліотек для побудови графіків, з одного боку, дає необхідні можливості, а, з іншого, вимагає завантаження власне бібліотек, які повинні надавати достатній функціонал як для створення графічного відображення, так і для надання користувачу можливості взаємодіяти з цим відображенням, що в свою чергу вимагає відносно значного розміру таких бібліотек, а також значної обчислювальної роботи, виконаної на стороні клієнта. В наслідок цього зростає час, необхідний на завантаження даних та час необхідний на власне рендерінг [4].

Проблема в тому, що JavaScript бібліотеки повинні містити надлишковий код, велика частина якого не буде використана для побудови практично кожної окремо взятої сторінки.

Отже, на стороні сервера необхідне попереднє очищення коду: без цього складно буде досягти швидкої реакції клієнта.

3. Ціль і завдання дослідження

Мета роботи – зменшити надлишковість програмного коду, що реалізує інтерактивність графіків за рахунок попередньої підготовки сторінки на стороні сервера.

Пропозиція полягає в тому, щоб використовувати інтерпретатор JavaScript на стороні сервера для попереднього створення варіанту сторінки, що буде показаний користувачу, та прибирання зайвого коду та HTML тегів.

4. Оцінка ефективності методу шляхом ручної обробки JavaScript коду

Для оцінки була взята типова сторінка. Офіційний приклад використання бібліотеки Flot [5] рис.

1. Бібліотека Flot була обрана для проведення оцінки, оскільки це бібліотека, що широко використовується для побудови графіків. А також, це бібліотека з відкритим вихідним кодом, що дозволить зробити необхідні зміни в коді.

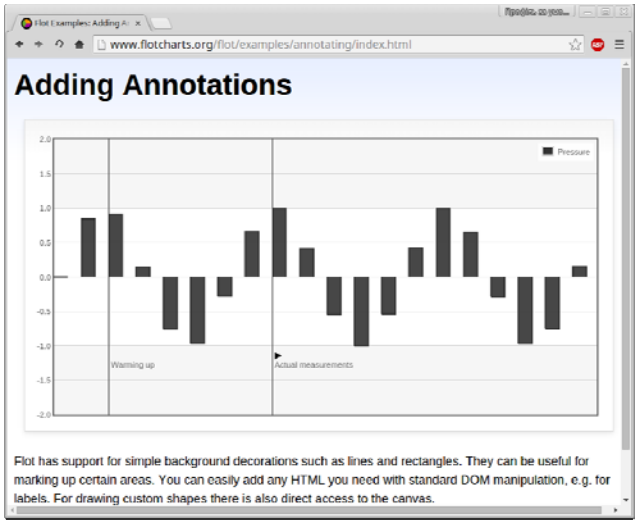

Рис. 1. Стовпчикова діаграма з аннотаціями

Початковий код даної сторінки був залишений без змін, а робота була здійснена над файлом jquery.flot.js, що містить код, який відповідає за відображення графіка.

Вручну було визначено ділянки коду, що не використовуються. Для цього в усіх функціях були поставлені мітки. В якості мітки було використано console.log(«$FUNCTION_NAME»), де $FUNCTION_NAME це назва функції в якій знаходиться мітка. Таким чином, був створений файл, що містить усі назви функцій, що були викликані. Також, таким же чином, були відмічені деякі блоки if та if/else.

В зв'язку з тим, що сторінка не буде змінюватись в майбутньому, необхідність в деяких перевірках відпадає.

В файлі з бібліотеки слово function зустрічається 132 рази. 3 рази в коментарях і 10 разів в допоміжному блоку, який не розглядався. Таким чином, бібліотека має 119 функцій. В кожній з них була розміщена мітка. Для побудови графіка було викликано 53 функції без врахування повторів. Отже, можна безпечно видалити 66 з них. Також, були видалені деякі перевірки.

5. Порівняння вихідної та модифікованої бібліотек

Файл бібліотеки до видалення містив 122971 символ. Модифікований файл містить 89522 символи. Отже, розмір скоротився приблизно на 28 відсотків.

Проте такі виміри не враховують той факт, що в бібліотеці наявні коментарі, а також символи, що не є необхідними для виконання кодом покладених на нього функцій, а потрібні лише для полегшення розуміння коду людиною, а також довгі назви функцій, з цією ж метою.

Для оптимізації був використаний сервіс JSCompress [6]. Оптимізований код початкової бібліотеки 36668 символів. Розмір бібліотеки з видаленими функціями, що не є необхідними для даної веб сторінки складає 25195 символів. Тобто розмір файлу скоротився на 32 відсотки.

Проте даний сервіс не скорочує назви функцій для того, щоб не руйнувати інтерфейс бібліотеки. Але, в нашому випадку, система повинна мати можливість робити таку оптимізацію. Також система повинна перевіряти всі розгалуження коду та видаляти ті, які не будуть використовуватись в документі, що дозволить ще більше збільшити виграш в скороченні об'єму даних, що завантажуються.

Тепер оцінимо виграш в часі. Для цього, було використано Google Drive [7] в якості хоста. Створимо таку ієрархію файлів:

```
.
├── annotating
│   └── index.html
├── examples.css
├── jquery.flot.js
└── jquery.flotO.js
```

Де index.html – це сам документ, examples.css – файл каскадної таблиці стилів, jquery.flot.js – файл, що містить бібліотеку з вирізаними зайвими функціями і jquery.flotO.js – файл, що містить оригінальний файл бібліотеки.

Для оцінки часу було використано Chrome DevTools.

Розмір оригінальної бібліотеки 31.3 KB. Розмір модифікованої бібліотеки 23.3 KB.

Було проведено 3 досліди. Результати дослідів наведені в табл. 1–3.

Таблиця 1

Час завантаження скрипта

№ експерименту	Модифікована бібліотека (с)	Оригінальна бібліотека (с)
1	494	759
2	347	439
3	394	180
Середнє	411.(6)	459.(3)

Таблиця 2

Час завантаження DOM

№ експерименту	Модифікована бібліотека (с)	Оригінальна бібліотека (с)
1	1.35	1.71
2	1.24	1.37
3	1.24	0.633
Середнє	1.27(6)	1.237(6)

Таблиця 3

Час завантаження документу

№ експерименту	Модифікована бібліотека (с)	Оригінальна бібліотека (с)
1	2.08	2.41
2	1.86	1.99
3	1.91	1.24
Середнє	1.95	1.88

Як видно на рис. 2 Хоча новий файл завантажується на 10.2 % швидше в порівнянні з початковим, це не вплинуло на швидкість завантаження документу. Після завантаження HTML-сторінки виконуються асинхронні запити на три файли: examples.css, jquery.js та jquery.flot.js. На час завантаження документу впливає запит, який виконується найдовше. Отже, система повинна мати змогу обробляти і бібліотеку jQuery. Без цього збільшення швидкості завантаження буде незначним. Потрібно також об'єднувати необхідні файли в один у випадку, якщо документ створюється один раз і потім не буде змінюватись.

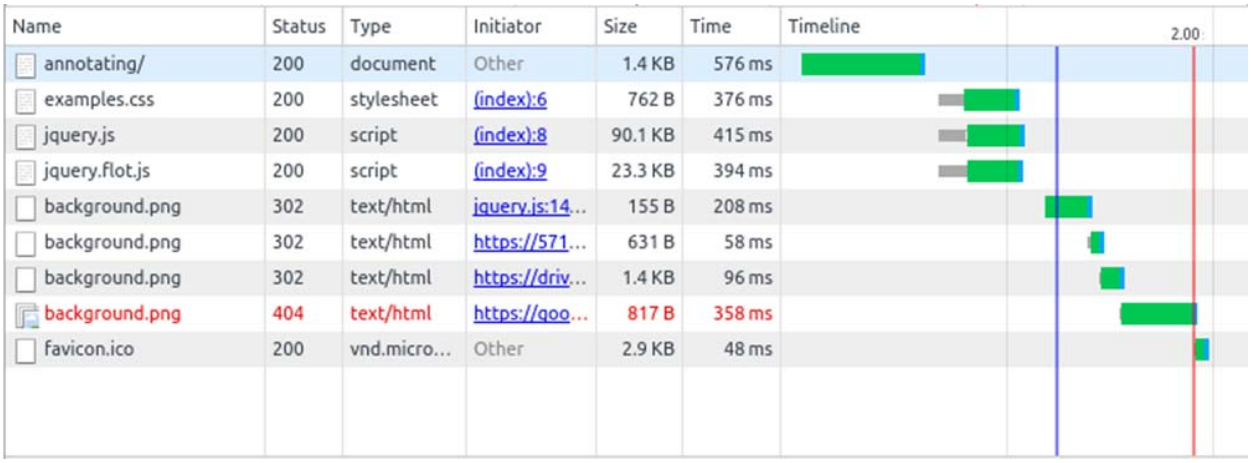

Рис. 2. Часова діаграма завантаження

6. Переваги та недоліки модифікованої бібліотеки

У зв'язку з наявністю виграшу в швидкості завантаження за рахунок відмови від універсальності та винесення спільного коду, така система буде найбільше підходити для випадку, коли користувач буде переглядати один документ, що буде містити однотипні графіки.

Така ситуація може виникнути у випадку використання певного типу конструктора або редактора документів, коли користувач має можливість додавати графіки, налаштовувати їх, додавати інтерактивність, а потім використовувати цей документ для звітів або презентацій. Тоді такий конструктор матиме певний набір бібліотек JavaScript і матиме усі необхідні умови, щоб надати користувачу саме мінімальний набір даних та коду для відображення документу. Кешування буде відбуватись в рамках одного документа, а універсальність в такому контексті, не вимагається.

Якщо це буде сайт з великою кількістю документів, які використовують можливості однієї бібліотеки, то така система втрачає сенс, оскільки файл бібліотеки просто буде кешуватись в браузері і в разі необхідності буде завантажений не з мережі, а з локального кешу для різних документів. Тому, загальний час завантаження кількох сторінок буде меншим.

Для, щоб виконати таку роботу автоматично, потрібно мати функціонал, аналогічний браузеру на стороні сервера. Повний чи майже повний функціонал браузера надають проекти:

PhantomJS – це заснований на рушії WebKit веб-браузер, без графічного інтерфейсу, що має JavaScript API. Він підтримує різні веб стандарти, такі як DOM handling, CSS selector, JSON, Canvas та SVG. Також має плагін для Node.js. [8].

HtmlUnit – браузер без графічного інтерфейсу, написаний на Java. Він надає програмам на Java можливість працювати з веб-сторінкою, має хорошу підтримку JavaScript, та може працювати зі складними AJAX бібліотеками. Здатний симулювати Firefox чи InternetExplorer залежно від конфігурації [9].

Також надають функціонал браузера, але не будують відображення DOM (через що підвищується швидкість роботи, проте знижується точність відображення сторінки):

Zombie.js – середовище симульованого браузера для Node.js. [10].

ENVJS – середовище симульованого браузера, написане на JavaScript для рушія Rino [11].

Існують бібліотеки, що дозволяють завантажити весь необхідний код та створити початковий варіант сторінки на стороні сервера. Необхідно лише створити систему, яка буде аналізувати, який код використовується, та прибирати зайвий.

Отже, якщо виконувати таку роботу на стороні сервера, то користувач отримає мінімальний необхідний набір даних, для відображення кінцевого результату.

Окрім того, що таке рішення пришвидшить завантаження сторінки без втрати інтерактивності, воно буде "бібліотеконезалежним", тобто таким, яке дозволить використовувати будь-яку наявну бібліотеку JavaScript, яка має хорошу модульну структуру.

Якщо включити код бібліотеки в документ, то буде необхідний лише один запит до сервера, але тоді втратиться перевага кешування, і для кожного нового документу необхідно буде завантажувати один і той же код заново. З іншого боку, можна залишити оброблений файл спільним для кількох документів. Проте, чим більшою буде їх різноманітність, тим менший буде виграш.

Таким чином алгоритм мінімазації даних буде мати вигляд рис. 3.

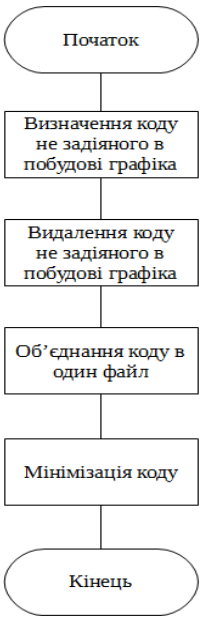

Рис. 3. Алгоритм обробки документу

7. Висновки

Запропоновано метод зменшення часу та об'єму даних, що передаються за рахунок попередньої обробки веб-сторінки на стороні сервера. Проведені експерименти показали, що застосування такого способу дозволяє скоротити об'єм файлу, який оброблюється на 30 %. Це дозволило знизити час, необхідний для завантаження файлу, на 10 %.

Використання цього способу не вплине на кінцевий вигляд сторінки. Тобто рішення дозволяє зберегти інтерактивність. Також запропонований метод є універсальним. Тобто таким, який буде працювати з будь-якою JavaScript бібліотекою в якій наявний код, що не є необхідним для побудови сторінки.

Найкраще метод працює в контексті одного документу, з відносною одноманітністю графіків та невисоким використанням функцій бібліотеки.

Недоліком способу є те, що якщо збільшується використання можливостей бібліотеки, знижується його ефективність в зв'язку зі зростанням кількості коду, що використовується.

Подальше вдосконалення способу буде направлене на зменшення цього недоліку за рахунок використання завантаження за вимогою (lazy loading).

Література

1. Welcome To JFreeChart! [Electronic Resource]. – Available at: http://www.jfree.org/jfreechart/

2. Let the computer in the cloud build your charts. [Electronic Resource]. – Available at: https://code.google.com/p/charts4j/

3. Attractive JavaScript plotting for jQuery [Electronic Resource]. – Available at: http://www.flotcharts.org/

4. A powerful and easy to use charting library for Ember.js. [Electronic Resource]. – Available at: http://addepar.github.io/ember-charts/#/ember-charts/overview

5. Flot examples: Adding Annotations [Electronic Resource]. – Available at: http://www.flotcharts.org/flot/examples/annotating/index.html

6. Online Javascript Compression Tool. [Electronic Resource]. – Available at: http://jscompress.com/

7. Усі ваші файли завжди під рукою [Electronic Resource]. – Available at: https://www.google.com/intl/uk/drive/

8. Full web stack no browser required. [Electronic Resource]. – Available at: http://phantomjs.org/

9. HtmlUnit. [Electronic Resource]. – Available at: http://htmlunit.sourceforge.net/

10. Insanely fast, headless full-stack testing using Node.js [Electronic Resource]. – Available at: http://zombie.js.org/

11. ENVJS – Brining the browser [Electronic Resource]. – Available at: http://www.envjs.com/

References

1. Welcome To JFreeChart! – Available at: http://www.jfree.org/jfreechart/

2. Let the computer in the cloud build your charts. – Available at: from https://code.google.com/p/charts4j/

3. Attractive JavaScript plotting for jQuery. – Available at: http://www.flotcharts.org/

4. A powerful and easy to use charting library for Ember.js. – Available at: http://addepar.github.io/ember-charts/#/ember-charts/overview

5. Flot examples: Adding Annotations. – Available at: http://www.flotcharts.org/flot/examples/annotating/index.html

6. Online Javascript Compression Tool. – Available at: http://jscompress.com/

7. A safe place for all your files. – Available at: https://www.google.com/intl/en/drive/

8. Full web stack no browser required. – Available at: http://phantomjs.org/

9. HtmlUnit. – Available at: http://htmlunit.sourceforge.net/

10. Insanely fast, headless full-stack testing using Node.js. – Available at: http://zombie.js.org/

11. ENVJS – Brining the browser. – Available at: http://www.envjs.com

Рекомендовано до публікації д-р техн. наук Стіренко С. Г.
Дата надходження рукопису 23.06.2015

Болдак Андрій Олександрович, кандидат технічних наук, доцент, кафедра обчислювальної техніки, Національний технічний університет України «Київський політехнічний інститут», пр. Перемоги, 37, Київ, Україна, 03056
E-mail: aboldak@comsys.kpi.ua

Мазуревич Юрій Вікторович, кафедра обчислювальної техніки, Національний технічний університет України «Київський політехнічний інститут», пр. Перемоги, 37, Київ, Україна, 03056
E-mail: jMazurevich@gmail.com