Т. 10. №12 2024
https://doi.org/10.33619/2414-2948/109

UDC 004.4 https://doi.org/10.33619/2414-2948/109/22

MEMORY LEAKS IN SPA: PREVENTION, DETECTION, AND REMEDIATION METHODS

© ***Dudak A.,*** *ORCID: 0009-0002-8466-8587, Tomsk State University of Control Systems and Radioelectronics, Tomsk, Russia, aleksei.dudak@rambler.ru*

УТЕЧКИ ПАМЯТИ В SPA: МЕТОДЫ ПРОФИЛАКТИКИ, ОБНАРУЖЕНИЯ И БОРЬБЫ С НИМИ

©Дудак А.А., *ORCID: 0009-0002-8466-8587, Томский государственный университет систем управления и радиоэлектроники, г. Томск, Россия, aleksei.dudak@rambler.ru*

Abstract. This article addresses the issue of memory leaks in modern single-page applications (SPAs). By investigating the causes of leaks associated with dynamic content updates, active interaction with the document object model (DOM) interface, and asynchronous operations, developers gain insights into avoiding the excessive accumulation of unused objects in memory. The article discusses methods for preventing and addressing leaks, including the use of weak references, component state management, and optimizing asynchronous requests. It also emphasizes the importance of using monitoring tools, such as Chrome DevTools, and integrating automated testing into the continuous integration (CI) and continuous delivery (CD) process. The article offers a comprehensive approach for efficient memory management and preventing performance issues in SPA applications.

Аннотация. В данной статье рассматривается проблема утечек памяти в современных одностраничных приложениях (SPA). Исследование причин утечек, связанных с динамическим обновлением контента, активным взаимодействием с программным интерфейсом DOM и асинхронными операциями помогает разработчикам понять, как избежать излишнего скопления неиспользуемых объектов в памяти. Рассматриваются методы профилактики и устранения утечек, включая использование слабых ссылок, управление состоянием компонентов, а также оптимизация работы с асинхронными запросами. Также акцентируется внимание на использовании инструментов мониторинга, таких как Chrome DevTools, и интеграции автоматизированного тестирования в процесс CI/CD. Статья предлагает комплексный подход для эффективного управления памятью и предотвращения проблем с производительностью в SPA-приложениях.

Keywords: memory leaks, single page applications (SPA), document object model (DOM), asynchronous operations, monitoring, optimization.

Ключевые слова: утечки памяти, одностраничные приложения (SPA), document object model (DOM), асинхронные операции, мониторинг, оптимизация.

Memory leaks are one of the common and complex issues faced by developers of modern single-page applications (SPAs). The key feature of such applications is that they dynamically update content without requiring page reloads, which leads to the retention of large amounts of data in the system’s memory. This approach enhances user experience, but if memory management is not handled correctly, unused objects may accumulate in the application, resulting in memory leaks.

Т. 10. №12 2024
https://doi.org/10.33619/2414-2948/109

The problem is further exacerbated by the complexity of state management in SPAs, where data can change throughout the application's lifecycle and interact frequently with various external data sources such as application programming interfaces (APIs) and databases.

Additionally, the nature of asynchronous requests and event handlers often leads to situations where references to objects are not deleted in time, leaving them in memory even though they are no longer needed. The prolonged accumulation of such objects can significantly degrade application performance and even cause crashes once the memory limit is reached. With the growing complexity of web applications and their increased interaction with users, memory leak issues are becoming increasingly critical. The aim of this study is to explore the factors contributing to memory leaks in SPAs and propose methods for their prevention and remediation. To achieve this, the causes of memory leaks, tools for their detection, and modern approaches to their prevention are examined.

Main part. Main characteristics and causes of memory leaks in SPA

Memory leaks in SPAs occur for several reasons related to the peculiarities of their architecture and handling dynamic data. Unlike traditional multi-page applications (MPAs), SPAs dynamically update content within a single page, which requires constant use of RAM to store data and UI elements. This characteristic, along with intensive interactions with the document object model (DOM) and asynchronous operations, increases the likelihood of memory leaks.

A memory leak, regardless of the architecture type, represents a serious challenge to the security and functionality of web applications. It can be exploited to disclose sensitive information, inject malicious code into the program's memory, and even be used in DoS (Denial of Service) attacks [1]. This is particularly concerning for DoS attacks, which, according to statistics, are increasing in scale (Figure). In the context of SPAs, memory leaks can be classified by their source types. One of the most common types is DOM element leaks, which occur when interface elements are dynamically created but not properly removed. For example, if the application adds nodes to the DOM based on user actions but fails to remove them when the state changes, those nodes continue to occupy memory even though they are no longer necessary [2].

Another type of leak involves closures. Closures, which are functions that have access to variables from an outer context, can hold references to objects even if they are no longer in use. In SPA, where asynchronous functions and event handlers are frequently used, poor management of closures can lead to the accumulation of references to outdated objects. Leaks caused by incomplete network requests and timers are also common. For instance, if an application creates multiple asynchronous requests or timers but does not stop them when the state changes, the resources associated with these operations remain in memory. This is especially significant in SPA, as such applications extensively use AJAX requests to exchange data with servers [3].

One of the primary causes of memory leaks in SPAs is their architectural approach, in which the interface is loaded once, and subsequent content changes happen without full page reloads. This approach ensures high interactivity but requires the constant retention of component states in memory. During operation, especially with frequent interface updates and significant state changes, memory may be unnecessarily occupied by unused elements and objects. The SPA approach heavily interacts with the DOM, and each DOM element occupies a significant amount of memory. In situations where interface elements are dynamically added or modified but not removed, even if they are no longer needed, this leads to the accumulation of unused DOM nodes that remain in memory. This issue worsens when DOM elements contain nested objects, such as images or multimedia files, as they occupy more memory and require additional resources. Moreover, SPA widely uses asynchronous operations and event handlers, increasing the likelihood of “dead”references”—situations where objects remain in memory despite no actual use. For example,

Т. 10. №12 2024
https://doi.org/10.33619/2414-2948/109

events attached to objects that have been removed from the DOM continue to exist if they were not explicitly removed. This results in associated objects continuing to occupy memory, causing unwanted leaks. Finally, it is important to note that memory leaks can also arise from unoptimized code and a lack of a well-thought-out state management architecture (Table).

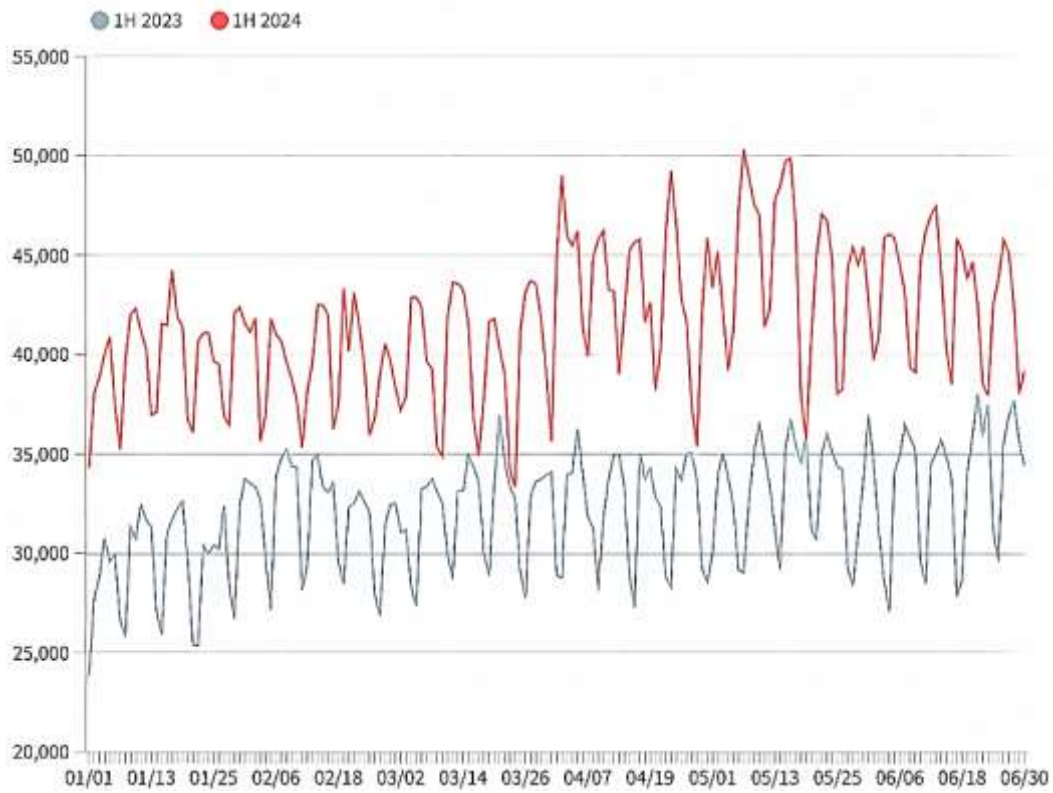

Figure. Volumetric DDoS Attacks 1H 2023 vs. 1H 2024 (https://lyl.su/o9Si)

Table

EXAMPLES OF COMMON MEMORY LEAKS IN SPA FRAMEWORKS
(https://lyl.su/Bcmg; https://lyl.su/U5hQ) [4]

Framework	*Typical leak*	*Cause*	*Remediation method*
React	Dom element leak	Incorrect state handling	Use useeffect for cleanup
Angular	Leak from asynchronous operations	Unfinished http requests	Use ngondestroy
Vue	Event leaks	Unremoved event handlers	Use beforedestroy to remove events

In the context of SPAs, state management is often handled using libraries such as React, Angular, and Vue, which store application state in memory. Without proper memory management, this can lead to the accumulation of outdated states, which continue to consume system resources. Additionally, unoptimized code, featuring bulky closures and unnecessary references to objects, can further exacerbate this issue.

Methods for resolving detected memory leaks

Eliminating memory leaks in SPAs requires systematic, technically sound methods. The resolution phase involves not only identifying and freeing up memory but also implementing measures to prevent their recurrence. It's essential to address DOM leaks, optimize closures, and handle asynchronous requests and event handlers appropriately.

Leaks DOM element occurs when nodes are dynamically created but not removed after they are no longer needed. To resolve this, developers should maintain strict control over the creation and deletion of DOM nodes. A common method is to manually remove unused nodes, freeing up

Т. 10. №12 2024
https://doi.org/10.33619/2414-2948/109

memory occupied by outdated UI elements. This can be achieved by using functions that both remove elements from the DOM and clear any references to them in the code.

Automatic DOM management can also be a helpful solution for SPAs, especially when working with a large number of dynamic objects. Modern frameworks like React and Vue support virtual DOM, allowing for automatic cleanup of removed or modified elements. However, developers must remember that even with a virtual DOM, memory leaks can occur if references are not managed properly. Therefore, it is important to use built-in cleanup methods and element removal techniques. For example, React's useEffect hook allows developers to configure automatic resource cleanup after they are no longer in use [5].

Closures, being an essential part of functional programming, often lead to memory leaks when captured variables remain in memory even when their values are no longer needed. To prevent and resolve closure-related leaks, it is recommended to remove references to closures when they are no longer required. An important method is the use of weak references, which allow developers to create references to objects without keeping them in memory, thus enabling the garbage collector to reclaim unused memory more effectively. In some cases, regular functions containing closures can be replaced by lighter callback functions, allowing memory to be freed immediately after the operation completes. For example, in asynchronous operations where closures use temporary variables, it’s crucial to ensure that function calls are correctly concluded after execution.

Another common source of memory leaks is event handlers that remain attached to objects even after those objects are removed from the DOM. This problem arises when event handlers are not detached when the component’s state changes or when an element is removed. To prevent and fix this, it is necessary to explicitly remove attached event handlers using the removeEventListener method in JavaScript. An example of such an approach is using React hooks, which provide automatic tracking of a component’s state. The useEffect hook can accept a cleanup function that removes all attached events on each component update, thereby preventing the accumulation of unused references. Other frameworks offer similar capabilities, utilizing built-in methods that track component state changes and perform automatic cleanup of handlers.

Memory leak monitoring and automation testing in SPA

One of the key steps in preventing memory leaks in SPA applications is continuous performance monitoring and automated testing. These approaches not only help identify memory leaks early in the development process but also ensure effective performance management in real time. It is essential for developers to use both analysis tools and approaches integrated into CI/CD (Continuous Integration and Delivery) systems to thoroughly check and automatically resolve memory leaks [6].

Modern browsers provide various tools for performance monitoring and detecting memory leaks. Chrome DevTools is one of the most powerful solutions for analyzing memory usage in web applications. With tools such as Heap Snapshots, Timeline Profiling, and Allocation Tracking, developers can track which objects remain in memory after the application finishes and identify objects that were not deleted in a timely manner. These tools help developers and testers pinpoint where memory leaks occur and which objects are causing them.

An example of using such tools is executing heap snapshots in Chrome DevTools, which provide detailed reports on which objects are consuming memory, how much memory has been allocated for each object, and how much memory was not freed after the operation finished. This data can be used to identify issues related to dynamically creating and deleting elements in the DOM, as well as analyzing the handling of asynchronous operations.

Memory leak monitoring and performance testing should be integrated into the CI/CD process to detect issues early in development. This reduces the likelihood of memory leaks being discovered

Т. 10. №12 2024
https://doi.org/10.33619/2414-2948/109

only during the application's production stage. Automation of testing, using tools like Jest, Mocha, and Cypress, can include creating tests that monitor memory usage and alert to deviations during test execution. In addition to these tools, performance testing libraries such as jest-memory-leak can run automatic memory leak checks during the testing process. These tests can be configured to monitor memory allocation with each code update, helping prevent situations where a memory leak goes unnoticed until the application is deployed.

State management libraries and tools

State management libraries like Redux, Vuex, and MobX are widely used in SPAs for centralized data storage. However, if state is not managed properly, data can accumulate in memory, causing leaks. It is important to integrate tools like Redux DevTools or Vue DevTools to track state changes and identify potential leaks.

Redux DevTools, for instance, allows developers to monitor all state changes in the application and shows which objects have been stored in memory. Vue DevTools similarly lets developers observe component states and dynamically changing data. These tools help identify leaks related to unnecessary references to data and allow developers to fix application behavior before deployment. One way to minimize memory leaks caused by asynchronous operations is by using «cancellation callbacks». These functions enable the cancellation or completion of operations related to network requests or timers once a component has been destroyed. This is crucial for preventing the accumulation of asynchronous operations that remain active even after the state changes. Integrating such methods into automated tests provides further assurance that the application will not have unused asynchronous processes lingering in memory, thus improving performance and reducing the likelihood of leaks in the production version.

Conclusion

Memory leaks in SPA applications present a significant and frequent issue that requires careful attention to design, development, and testing. SPA architectural features, such as dynamic content updates and the heavy use of asynchronous operations, increase the likelihood of memory leaks. However, with the right memory management methods, monitoring, and automated testing, these issues can be effectively prevented.

Key preventive measures include using modern monitoring tools like Chrome DevTools and integrating memory testing into the CI/CD process, which helps identify leaks early in development. Additionally, it is crucial to manage component states correctly, use weak references, and terminate asynchronous operations when they are no longer needed.

Substantial attention should be paid to training the development team and employing code review practices to minimize the chance of memory leaks. Continuous monitoring and the use of memory testing tools should become an integral part of the workflow, ensuring the long-term stability and performance of SPA applications.

References:

1. Israfilov Anar (2024). Geopolitical aspects of cybersecurity: international cooperation and conflicts. *Kholodnaya nauka*, (8), 56-63.

2. Shahoor, A., Khamit, A. Y., Yi, J., & Kim, D. (2023). LeakPair: Proactive repairing of memory leaks in single page web applications. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 1175-1187). IEEE. https://doi.org/10.1109/ASE56229.2023.00097

3. Yu, B., Tian, C., Zhang, N., Duan, Z., & Du, H. (2021). A dynamic approach to detecting, eliminating and fixing memory leaks. *Journal of Combinatorial Optimization*, *42*, 409-426. https://doi.org/10.1007/s10878-019-00398-x

4. Utture, A., & Palsberg, J. (2023, November). From Leaks to Fixes: Automated Repairs for Resource Leak Warnings. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 159-171). https://doi.org/10.1145/3611643.3616267

5. Ponomarev, E. V. (2024). Razrabotka kreditnykh prilozhenii na Android: osobennosti i vyzovy. *Vestnik nauki*, *2*(9 (78)), 319-327. (in Russian).

6. Cák, F., & Dakić, P. (2024, September). Configuration Tool for CI/CD Pipelines and React Web Apps. In *2024 14th International Conference on Advanced Computer Information Technologies (ACIT)* (pp. 586-591). IEEE. https://doi.org/10.1109/ACIT62333.2024.10712482

Список литературы:

1. Israfilov A. Geopolitical aspects of cybersecurity: international cooperation and conflicts // Холодная наука. 2024. №8. P. 56-63.

2. Shahoor A., Khamit A. Y., Yi J., Kim D. LeakPair: Proactive repairing of memory leaks in single page web applications // 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2023. P. 1175-1187. https://doi.org/10.1109/ASE56229.2023.00097

3. Yu B., Tian C., Zhang N., Duan Z., Du H. A dynamic approach to detecting, eliminating and fixing memory leaks // Journal of Combinatorial Optimization. 2021. V. 42. P. 409-426. https://doi.org/10.1007/s10878-019-00398-x

4. Utture A., Palsberg J. From Leaks to Fixes: Automated Repairs for Resource Leak Warnings // Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2023. P. 159-171. https://doi.org/10.1145/3611643.3616267

5. Пономарёв Е. В. Разработка кредитных приложений на Android: особенности и вызовы // Вестник науки. 2024. Т. 2. №9 (78). С. 319-327.

6. Cák F., Dakić P. Configuration Tool for CI/CD Pipelines and React Web Apps // 2024 14th International Conference on Advanced Computer Information Technologies (ACIT). IEEE, 2024. P. 586-591. https://doi.org/10.1109/ACIT62333.2024.10712482

Работа поступила
в редакцию 06.11.2024 г.

Принята к публикации
12.11.2024 г.

Ссылка для цитирования:

Dudak A. Memory Leaks in Spa: Prevention, Detection, and Remediation Methods // Бюллетень науки и практики. 2024. Т. 10. №12. С. 161-166. https://doi.org/10.33619/2414-2948/109/22

Cite as (APA):

Dudak, A. (2024). Memory Leaks in Spa: Prevention, Detection, and Remediation Methods. *Bulletin of Science and Practice*, *10*(12), 161-166. https://doi.org/10.33619/2414-2948/109/22