World Journal of Innovation and Modern Technology, Vol. 7, Issue 6, (Dec) 2024
ISSN 2682-5910
DOI: 10.53469/wjimt.2024.07(06).07

Research and Application of Asynchronous Programming in JavaScript

Weizhi Liu, Lin Li

School of Computer and Software, Jincheng College, Sichuan University, Chengdu 611731, China

Abstract: ***JavaScript, as a widely used scripting language in web development, did not initially include direct support for concurrent execution of multiple tasks in its design. This means that in the traditional JavaScript programming model, tasks need to be queued and executed in order, greatly limiting the efficiency and responsiveness of the program, especially when dealing with situations such as network requests, file reads and writes, or user interactions that require waiting for external resources. To address this issue, asynchronous programming patterns have emerged, allowing programs to continue executing other tasks while waiting for certain operations to complete, thereby avoiding task blocking and queuing. The core idea of asynchronous programming lies in non blocking operations, which means that when a task (such as a network request) starts, it does not block the execution of subsequent code, but immediately returns and processes the result through callback functions after the task is completed. This mode greatly improves the response speed and user experience of applications, especially in complex web applications. With the release of ECMAScript 6 (ES6 for short), JavaScript took an important step in the field of asynchronous programming by introducing Promise objects. Promise is an object that represents the final completion or failure of an asynchronous operation and carries the value of the operation result. Its emergence provides a clearer and more chained way for asynchronous programming to handle asynchronous tasks and their results, effectively avoiding the common "callback hell" problem in traditional callback functions. This article delves into the application of Promise objects in JavaScript asynchronous programming. Promise not only simplifies the structure of asynchronous code, making asynchronous logic more intuitive and understandable, but also provides powerful error handling mechanisms through its. then(),. match(), and. terminal() method chains. In addition, Promise supports parallel execution of multiple asynchronous operations and achieves finer grained control over multiple asynchronous tasks through methods such as Promise. all() and Promise. race(). The introduction of Promise objects marks a significant advancement in asynchronous programming in JavaScript. It not only improves code readability and maintainability, but also provides strong support for developers to build high-performance and highly responsive web applications. With the continuous development of technology, promises have become an indispensable part of modern JavaScript development.***

Keywords: Asynchronous programming; Promise object; Async/await.

1. INTRODUCTION

JavaScript, originally designed as a scripting language solely for the browser side, has gradually crossed its original boundaries with the rapid development of web technology and the increasing demand for more efficient and flexible programming [1]. It has become a full stack language that can serve both front-end browsers and back-end server development. This evolution process not only witnessed the continuous expansion and strengthening of JavaScript's own functions, but also profoundly influenced the methods and concepts of modern software development [2]. Early JavaScript was mainly used to enhance the interactivity of web pages, such as form validation, dynamic content display, etc. However, with the rise of Ajax technology, JavaScript has begun to demonstrate its enormous potential in handling asynchronous requests and achieving refresh free page updates [3] [4] [5]. This breakthrough marks the official entry of JavaScript into the field of asynchronous programming, laying the foundation for its widespread application in more complex scenarios in the future. Subsequently, the emergence of Node.js completely changed the ecological landscape of JavaScript. Node.js is a JavaScript runtime built on the Chrome V8 engine that enables JavaScript to run on the server-side, handling tasks such as HTTP requests, file system operations, database interactions, and more traditionally done by backend languages such as Java, PHP, Ruby, etc [6] [7] [8] [9]. This innovation not only greatly expands the application scope of JavaScript, but also promotes the trend of front-end and back-end unified language development, simplifies the development process, and improves development efficiency.

1.1 Single threaded and asynchronous programming: The way to optimize JavaScript performance

The single threaded nature of JavaScript was once considered a bottleneck in handling multitasking. However, it is precisely this feature that has prompted developers to explore asynchronous programming as an efficient way of handling tasks. In the single threaded model, JavaScript implements non blocking handling of asynchronous tasks through the Event Loop mechanism. This means that when JavaScript code encounters waiting operations such as

network requests, timer settings, etc., it will not pause the execution of the entire program, but rather suspend these tasks and continue to execute subsequent code [10]. Once the asynchronous task is completed, the corresponding callback function will be pushed into the event queue and wait for the main thread to idle for execution. The introduction of asynchronous programming has greatly improved the task execution efficiency of JavaScript, enabling it to better cope with high concurrency scenarios such as real-time communication and extensive user interaction. In addition, it also promotes the enrichment and development of JavaScript asynchronous APIs, such as Promise, asynchronous/await, etc [11] [12] [13]. These features make asynchronous code writing more concise and intuitive, reduce error rates, and improve code maintainability.

1.2 Deepening and Expanding Asynchronous Programming

With the release of ES6 (ECMAScript 2015) and subsequent versions, JavaScript has made significant progress in asynchronous programming [14]. The introduction of Promise objects provides a more elegant and reliable way to handle the results of asynchronous operations. Promise represents the final completion (or failure) of an asynchronous operation and its resulting value. It allows developers to organize asynchronous code in a chained manner, avoiding common issues such as deep nesting and complex error handling in traditional callback functions. Furthermore, the emergence of the asynchronous/await syntax has pushed asynchronous programming to a new level [15] [16] [17] [18]. By marking asynchronous functions as asynchronous and using the await keyword inside the function to wait for Promise resolution, developers can write asynchronous logic that looks almost identical to synchronous code, greatly enhancing the readability and comprehensibility of the code. With the widespread application of JavaScript in the front-end and back-end fields, the task requirements it faces are becoming increasingly complex and diverse. In order to better handle these requirements, JavaScript constantly absorbs new features and strengthens its own capabilities. For example, introducing modular mechanisms such as ES6 modules, CommonJS, etc., makes the organization and management of code clearer and more efficient; Supporting classes and inheritance enhances the ability of object-oriented programming; Providing syntax sugar such as template strings and deconstruction assignments simplifies the implementation of common programming patterns. In addition, JavaScript also explores the possibility of implementing multi-threaded parallel processing in browser environments through technologies such as Web Workers and Service Workers, in order to further improve performance and response speed [19] [20] [21] [22]. Although these technologies are currently mainly used to handle specific tasks such as backend data processing, offline caching, etc., they provide useful attempts and ideas for JavaScript to achieve more efficient concurrent processing in the future.

JavaScript has evolved from a simple browser scripting language to a full stack language capable of front-end and back-end development. The continuous updating and strengthening of its functionality, especially the introduction and deepening of asynchronous programming features, have played a crucial role. With the continuous advancement of technology and the expansion of application scenarios, JavaScript will continue to evolve to meet various task requirements in a more efficient and flexible way, promoting the development and innovation of web technology [23] [24].

2. INTRODUCTION TO JAVASCRIPT ASYNCHRONOUS MODE

In JavaScript, due to the single threaded execution environment, JavaScript can only execute one task at a time, and subsequent tasks can only be queued for waiting. This will prevent the task to be executed from being executed immediately, affecting the efficiency of task execution. The main purpose of JavaScript language for browsers is to render DOM, and only single threaded mode can avoid multiple segments of DOM modification operations at the same time [25]. Therefore, the JavaScript language has chosen the single threaded mode. Although this mode can make its operating environment relatively simple and easier to implement; But the disadvantage is that when multiple tasks are being performed simultaneously, JavaScript will only cause these tasks to queue and wait, and will only continue to execute the following tasks if the previous one is completed first. So, in order to solve the problem of multiple tasks not being able to be performed simultaneously. The asynchronous programming pattern in JavaScript emerged from this.

When JavaScript performs asynchronous programming, all tasks first wait to be executed on the main thread. When starting work, generate heap and stack [26]. In the execution stack, asynchronous tasks will call the corresponding API externally, and then add various different events to the message queue. After the synchronization task in the main thread is completed, it will continue to loop through the message queue to retrieve these events and execute them. Completing such a loop is called an event loop, and the main thread will continue to

perform such an event loop. These called events are executed in the main thread, not in the worker thread. Therefore, this working mode ensures that JavaScript processes events in a way that prevents them from blocking.

There are four main methods of asynchronous programming: (1) Callback function. (2) Event monitoring. (3) Publish subscriptions. (4) Promise object. This article mainly studies the fourth type: Promise object.

3. RESEARCH ON PROMISE OBJECTS

Before introducing the promise object, this article first introduces two concepts: macro tasks and micro tasks. In JavaScript, macro tasks have setTimeout and setInterval methods; Micro tasks have promise and process.nextTick methods. The execution order of JavaScript is to execute code line by line from top to bottom. When encountering macro and micro tasks, the micro tasks will be placed in the micro task queue, and the macro tasks will be placed in the macro task queue. Firstly, loop the micro task queue to execute and output all the micro tasks. When the micro task queue is empty, loop the macro task queue to output the macro tasks. The end flag of an event loop is when all macro tasks have been executed and the micro task queue is empty with no micro tasks yet to be executed.

According to the Promise/A specification, a promise is an object used to handle asynchronous operations. It itself has methods such as resolve and reject, and the prototype has methods such as then and catch. Promise objects have three states: pending, resolved, and rejected. Pending means that the promise is in progress and has not yet been completed. When the status is resolved, it indicates that the promise is in a completed state; When the status is rejected, it indicates that the promise is in a failed state. The state of a Promise is just like its name implies, only the resolve and reject functions can change its state. The resolve function and reject function are methods to change the promise state. Resolve changes the state of a promise from pending to fulfilled, while reject changes it from pending to rejected. When the state of a promise changes, it will not change again, and the states between resolved and rejected will not change from each other.

Promise has two prototype methods. The first prototype method is the then method. It has two parameters, one is the callback of the resolve function, and the other is the callback of the reject function, the latter is optional. The second prototype method is the catch method, which only accepts callbacks from the reject function. When the state of the promise is rejected, the catch method will accept the callback and output it. Usually, when using the then method, only the first parameter is set, which means only resolved callbacks are accepted. The catch method is usually used to accept rejected callbacks. For promises, chain calls are their most prominent part. Since the then method returns a new promise object, it can be continuously called like a chain. In addition, a catch method can be added at the end of the then method in the chain call, which is used to accept rejected callbacks and handle errors that occur during runtime. However, it should be noted that if the previous' then 'method returns a rejected callback, the subsequent' then 'method will not be called, but will jump directly to the catch method at the end.

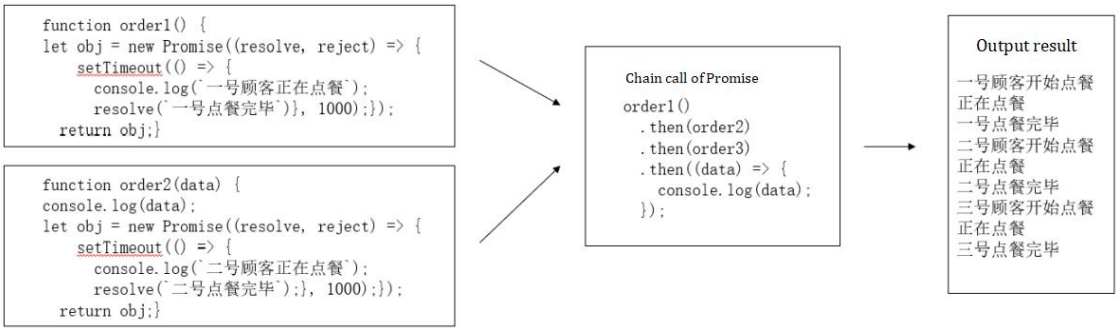

Figure 1: Example of Promise Chain Call

As shown in the code in Figure 1, first define three functions, and the format of order3 is consistent with order2. Both order2 and order3 functions have added a data parameter, which is the resolve function parameter that accepts the promise object returned by the previous function. A promise object is created in each function and given two parameters to receive the resolve and reject functions. The setTimeout method is used inside the function to output and pass the value to the resolve function. When calling them in a chain, the value of the previous resolve function is passed to the data parameter of the next function, and the output result is shown at the far right in the figure.

With the introduction of ES2017, a brand new asynchronous function has been introduced. This function, as a syntactic sugar, is also based on the implementation of promises. The use of the Asynchronous function is to prefix the original synchronous function with the 'asynchronous' keyword, making it an asynchronous function. This function is usually used in conjunction with await, and await cannot be used alone without an asynchronous declaration function. Await returns a promise object. When executing an asynchronous function, if there is an await keyword inside the function, it will first wait and start running the code after awaiting, and then return a promise object. Only after completion will the subsequent tasks continue. If the asynchronous operation of await is successful, the state of the promise will be changed to resolved. If it fails, the state will be changed to rejected, and the subsequent operations will be determined based on its state.

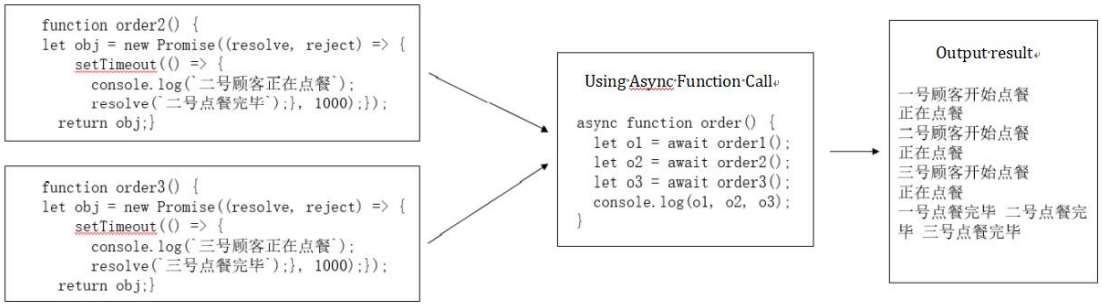

Figure 2: Example of Asynchronous Function Call

As shown in Figure 2, the code demonstrates an example modified with the use of the asynchronous function. The call to a promise object by the asynchronous function appears more standardized and tidy. Since await returns a promise object, the resolved state information returned by each promise object no longer needs to be received by the next function through parameters, but will be output together at the end. Therefore, the function in the figure does not require parameters and cancels the output of parameters. Output through the asynchronous function. This not only reduces the complexity of the function code, but also better presents the results after the function call.

4. APPLICATION OF PROMISE OBJECT

Due to Promise providing a unified paradigm and API for asynchronous programming, it has a wide range of application scenarios. Taking Axios, which the author used in the epidemic prevention and control management system, as an example. Axios is an AJAX technology that is also a promise based asynchronous access technology. In this project, the first step is to encapsulate an Axios method that accepts data uploaded from the frontend interface, matches the data with backend data, verifies it, and proceeds to the next step. The front-end page uses the encapsulated Axios method to pass user input information to the back-end for processing, achieving the connection between the front-end and back-end.

let Axios = (options) => { axios({url: options.url, method: options.method || 'POST', data: options.data, params: options.data,}). then((result) => { if (options. success) { options. Success (result.data); }}). catch((err) => { let msg = err.response ? err.response.data: '请求异常';if (options.error) { options.error(msg); essage.error({ message: msg, offset: 150 });} else {Message.error({ message: msg, offset: 150 });}});};

This code shows the code that encapsulates the axios method in the main. js file of the Vue project. Firstly, the axios method receives information passed from the frontend. As axios is a promise based method, it is subsequently processed through chain calls using the then and catch methods to handle different states. In this project, the obtained information is first evaluated and the result is encapsulated into a promise object, which is passed through a simple chain call. The next method to be called is determined by the state of the promise object. Traditional AJAX requests require nested code layers when making multiple sequential requests to the server, resulting in a callback hell. Since Axios is a promise based method, it is possible to avoid callback hell through chain calls when making sequential requests in Axios. This is also one of the main reasons for using promises. When sequentially querying the information input by the user, Axios encapsulates the results into a promise object. When a user logs in with a username and password, Axios retrieves the information entered by the user and encapsulates the results by checking if the user information matches. If correct, it will be encapsulated into a promise object with a resolved state, and the then method will accept the return value and perform the

corresponding operation. Otherwise, it will be encapsulated as an object with a rejected state, which will be received and called by the catch method.

The advantage of using promises to encapsulate methods in Axios is that it reduces the need for repetitive code writing and minimizes the layers of code nesting to avoid callback hell. The Axios method encapsulates the result as a promise object, and then and catch methods call the corresponding results of successful and failed operations. Another advantage is that it is easy to maintain and manage the code. For traditional AJAX, as the amount of code increases and the number of nested layers increases, it is not conducive to checking and correcting the code when errors occur. After Promise simplifies the code, it facilitates developers to accurately locate errors in the project, thereby reducing the number of error checking steps for the code and facilitating maintenance and management.

5. CONCLUSION

The emergence of Promise objects makes JavaScript's asynchronous processing of tasks more convenient and efficient. Compared to traditional callback functions and other asynchronous methods, the appearance of Promise improves the code that originally needed to be nested layer by layer, preventing the occurrence of callback hell and visualizing the execution process. Through chain calls, errors can be detected in a timely manner and changes and maintenance can be made. However, the chain call of promises may also result in callback hell, and when there is a large amount of data processing, chain calls can make the code appear complex. In the application scenarios of concurrent and sequential tasks, promises can handle such tasks well. And due to the emergence of the asynchronous function in ES7, most of the shortcomings and problems of promises in chain calls have been optimized. Therefore, when dealing with asynchronous tasks, promise objects are still the preferred choice.

REFERENCES

[1] Z. Ren, "A Novel Feature Fusion-Based and Complex Contextual Model for Smoking Detection," 2024 6th International Conference on Communications, Information System and Computer Engineering (CISCE), Guangzhou, China, 2024, pp. 1181-1185, doi: 10.1109/CISCE62493.2024.10653351.
[2] Teller, & Virginia. (2000). Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition daniel jurafsky and james h. martin (university of colorado, boulder) upper saddle river, nj: prentice hall (prentice hall ser. Computational Linguistics, 26(4), 638-641.
[3] Lin, Z., Wang, Z., Zhu, Y., Li, Z., & Qin, H. (2024). Text Sentiment Detection and Classification Based on Integrated Learning Algorithm. Applied Science and Engineering Journal for Advanced Research, 3(3), 27-33.
[4] He, C., Liu, M., Zhang, Y., Wang, Z., Hsiang, S. M., Chen, G., Li, W., & Dai, G. (2023). Space – Time – Workforce Visualization and Conditional Capacity Synthesis in Uncertainty. Journal of Management in Engineering, 39(2), 04022071. https://doi.org/10.1061/JMENEA.MEENG-4991
[5] He, C., Yu, B., Liu, M., Guo, L., Tian, L., & Huang, J. (2024). Utilizing Large Language Models to Illustrate Constraints for Construction Planning. Buildings, 14(8), 2511. https://doi.org/https://doi.org/10.3390/buildings14082511
[6] Nadkarni, P. M. , Ohno-Machado, L. , & Chapman, W. W. . (2011). Natural language processing: an introduction. Journal of the American Medical Informatics Association Jamia, 18(5), 544.
[7] Tian, Q., Wang, Z., Cui, X. Improved Unet brain tumor image segmentation based on GSConv module and ECA attention mechanism. arXiv preprint arXiv:2409.13626.
[8] Xu Y, Shan X, Guo M, Gao W, Lin Y-S. Design and Application of Experience Management Tools from the Perspective of Customer Perceived Value: A Study on the Electric Vehicle Market. World Electric Vehicle Journal. 2024; 15(8):378. https://doi.org/10.3390/wevj15080378
[9] Tennant, & Harry. (1981). Natural language processing: an introduction to an emerging technology.
[10] He, C., Liu, M., Wang, Z., Chen, G., Zhang, Y., & Hsiang, S. M. (2022). Facilitating Smart Contract in Project Scheduling under Uncertainty — A Choquet Integral Approach. Construction Research Congress 2022, 930 – 939. https://doi.org/10.1061/9780784483961.097
[11] Wyard, P. J. . (1992). Connectionist natural language processing: an introduction. Springer Netherlands.
[12] Wang, Zeyu. "CausalBench: A Comprehensive Benchmark for Evaluating Causal Reasoning Capabilities of Large Language Models." Proceedings of the 10th SIGHAN Workshop on Chinese Language Processing (SIGHAN-10). 2024.
[13] Liu, S., Li, X., & He, C. (2021). Study on dynamic influence of passenger flow on intelligent bus travel service model. Transport, 36(1), 25 – 37. https://doi.org/10.3846/transport.2021.14343

[14] Bethard, S. , Jurafsky, D. , & Martin, J. H. . (2008). Instructor's Solution Manual for Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition (Second Edition).
[15] Yao, J. (2024). The Impact of Large Interest Rate Differentials between China and the US bn the Role of Chinese Monetary Policy -- Based on Data Model Analysis. Frontiers in Economics and Management, 5(8), 243-251.
[16] Xu, Y., Gao, W., Wang, Y., Shan , X., & Lin, Y.-S. (2024). Enhancing user experience and trust in advanced LLM-based conversational agents. Computing and Artificial Intelligence, 2(2), 1467. https://doi.org/10.59400/cai.v2i2.1467
[17] Zheng, H., Wang, B., Xiao, M., Qin, H., Wu, Z., & Tan, L. (2024). Adaptive Friction in Deep Learning: Enhancing Optimizers with Sigmoid and Tanh Function. arXiv preprint arXiv:2408.11839.
[18] He, C., Liu, M., Zhang, Y., Wang, Z., Simon, M. H., Chen, G., & Chen, J. (2022). Exploit Social Distancing in Construction Scheduling: Visualize and Optimize Space – Time – Workforce Tradeoff. Journal of Management in Engineering, 38(4), 4022027. https://doi.org/10.1061/(ASCE)ME.1943-5479.0001037
[19] Wang, Z., Zhu, Y., Li, Z., Wang, Z., Qin, H., & Liu, X. (2024). Graph neural network recommendation system for football formation. Applied Science and Biotechnology Journal for Advanced Research, 3(3), 33-39.
[20] Yin, Y., Xu, G., Xie, Y., Luo, Y., Wei, Z., & Li, Z. (2024). Utilizing Deep Learning for Crystal System Classification in Lithium - Ion Batteries. Journal of Theory and Practice of Engineering Science, 4(03), 199 – 206. https://doi.org/10.53469/jtpes.2024.04(03).19
[21] Luo, Y., Wei, Z., Xu, G., Li, Z., Xie, Y., & Yin, Y. (2024). Enhancing E-commerce Chatbots with Falcon-7B and 16-bit Full Quantization. Journal of Theory and Practice of Engineering Science, 4(02), 52 – 57. https://doi.org/10.53469/jtpes.2024.04(02).08
[22] Jurafsky, D. , & Martin, J. H. . (2007). Speech and language processing: an introduction to speech recognition, computational linguistics and natural language processing. Prentice Hall PTR.
[23] Wang, Z., Sun, W., Chu, Z. C., Zhang, Y., & Wu, Z. (2024). LLM for Differentiable Surface Sampling for Masked Modeling on Point Clouds.
[24] Xie, Y., Li, Z., Yin, Y., Wei, Z., Xu, G., & Luo, Y. (2024). Advancing Legal Citation Text Classification A Conv1D-Based Approach for Multi-Class Classification. Journal of Theory and Practice of Engineering Science, 4(02), 15 – 22. https://doi.org/10.53469/jtpes.2024.04(02).03
[25] Xu, G., Xie, Y., Luo, Y., Yin, Y., Li, Z., & Wei, Z. (2024). Advancing Automated Surveillance: Real-Time Detection of Crown-of-Thorns Starfish via YOLOv5 Deep Learning. Journal of Theory and Practice of Engineering Science, 4(06), 1 – 10. https://doi.org/10.53469/jtpes.2024.04(06).01
[26] Z. Ren, "Enhancing Seq2Seq Models for Role-Oriented Dialogue Summary Generation Through Adaptive Feature Weighting and Dynamic Statistical Conditioninge," 2024 6th International Conference on Communications, Information System and Computer Engineering (CISCE), Guangzhou, China, 2024, pp. 497-501, doi: 10.1109/CISCE62493.2024.10653360.