Research and Application of Asynchronous Programming in JavaScript

Weizhi Liu[1], Lin Li[2]

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

Abstract: ***Due to the fact that JavaScript cannot perform multiple tasks simultaneously in programming, the emergence of asynchronous programming has made these tasks no longer require queuing, but can be performed asynchronously. And with the introduction of Promise objects in ES6, JavaScript has made further improvements in asynchronous programming. This article discusses the Promise object in this regard.***

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

1. INTRODUCTION

As JavaScript has evolved from being a scripting language for browsers to still serving backend servers, its functionality has been constantly updated and strengthened. Single threaded JavaScript, with its asynchronous programming capabilities, increases task execution efficiency and enables it to better handle a variety of task requirements and implementations.

The recent research applying artificial intelligence (AI) and machine learning (ML) techniques to various fields, including image processing, natural language processing (NLP), e-commerce, healthcare, and finance. The studies span different approaches and model architectures, highlighting the versatility and expanding capabilities of these technologies.

Image Processing and Computer Vision: Several papers explore advancements in image processing using deep learning. Yan et al. (2024) investigate image super-resolution reconstruction mechanisms based on convolutional neural networks. Tian et al. (2024) propose an improved U-Net model for brain tumor image segmentation, incorporating GSConv and ECA attention mechanisms. Ren (2024) presents an enhanced YOLOv8 approach for infrared image object detection, focusing on improving performance with the SPD module. These studies showcase the effectiveness of deep learning in enhancing image quality and enabling accurate object detection in various contexts.

Natural Language Processing (NLP) and Large Language Models (LLMs): The application of NLP and LLMs is a major theme, with several studies focusing on chatbot improvement and LLMs in general. Lu (2024) uses a machine learning approach integrating decision trees, TF-IDF, and BERTopic to enhance chatbot user satisfaction. Luo et al. (2024) explore enhancing e-commerce chatbots using Falcon-7B and 16-bit full quantization, aiming to improve efficiency and performance. Xu et al. (2024) focus on enhancing user experience and trust in advanced LLM-based conversational agents. Wu (2024) presents work on using LLMs for semantic parsing in an intelligent database query engine. Li et al. (2024) explores strategic deductive reasoning in LLMs using a dual-agent approach. Ren (2024) introduces a novel feature fusion-based model for smoking detection. The foundational work of Nadkarni et al. (2011), Bethard et al. (2008), Teller (2000) and Jurafsky & Martin (2007) provides context for the advancements in NLP.

E-commerce and Recommendation Systems: Li (2024) focuses on harnessing multimodal data and multi-recall strategies for enhanced product recommendations in e-commerce. This highlights the ongoing efforts to improve personalization and efficiency in online shopping experiences. Xu et al. (2024) investigate experience management tools in the electric vehicle market, relating to broader customer experience enhancement in digital settings including e-commerce.

Healthcare and Medical Applications: Lin et al. (2024a) provide a comprehensive review of precision anesthesia in high-risk surgical patients. Lin et al. (2024b) further explores innovative methods for optimizing anesthesia depth using AI and EEG analysis. These studies demonstrate AI's growing role in improving patient safety and surgical outcomes.

Finance and Risk Management: Bi et al. (2024) present research and design for a financial intelligent risk control platform using big data analysis and deep machine learning. Li et al. (2024) investigate the impact of policies promoting the integration of technology and finance on green innovation. These papers highlight the increasing use of AI and ML in financial technology for risk mitigation and strategic decision-making.

Other Applications: Liang and Chen (2019a, 2019b) explore network security and service orchestration in hybrid NFV networks. Qi and Liu (2024) develop a sales forecasting system using Hadoop big data analysis. Zheng et al. (2024) focus on adaptive friction in deep learning optimizers, suggesting improvements in model training. Yin et al. (2024) apply deep learning for crystal system classification in lithium-ion batteries. Lian and Chen (2024) and Chen et al. (2024) explore complex data mining and pattern recognition using deep learning techniques. Chen and Bian (2019) detail a streaming media live broadcast system using MSE.

It shows the breadth of AI and ML applications across various domains. While many studies focus on improving existing techniques, others explore new applications and architectures, indicating a continued evolution and expansion of these technologies. Further research should focus on addressing ethical considerations and ensuring responsible development and deployment of AI systems.

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. 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. 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.

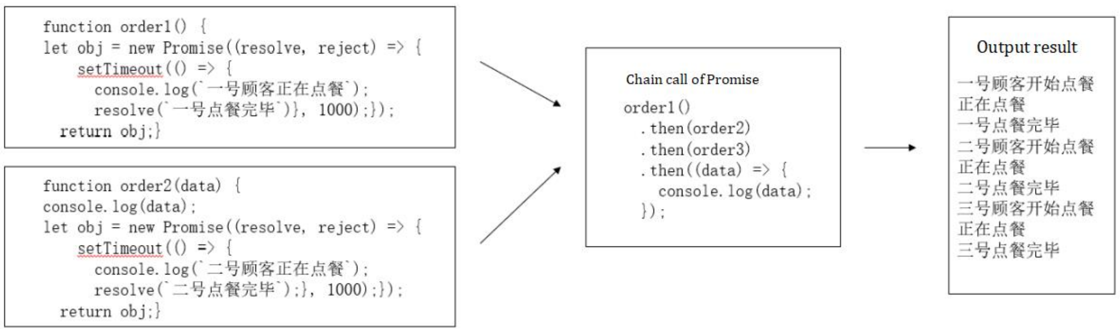
There are four main methods of asynchronous programming: (1) Callback function. (2) Event monitoring. (3) Publish subscriptions. (4) Promise object. This article mainly studies the fourth type: Promise object.

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.



```
function order1() {
let obj = new Promise((resolve, reject) => {
    setTimeout(() => {
      console.log(`一号顾客正在点餐`);
      resolve(`一号点餐完毕`)}, 1000);});
  return obj;}
```

```
function order2(data) {
console.log(data);
let obj = new Promise((resolve, reject) => {
    setTimeout(() => {
      console.log(`二号顾客正在点餐`);
      resolve(`二号点餐完毕`);}, 1000);});
  return obj;}
```

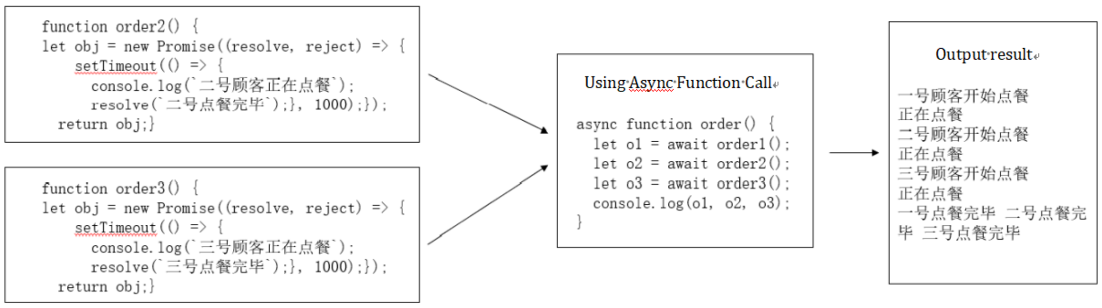
```
order1()
  .then(order2)
  .then(order3)
  .then((data) => {
    console.log(data);
  });
```

```
一号顾客开始点餐
正在点餐
一号点餐完毕
二号顾客开始点餐
正在点餐
二号点餐完毕
三号顾客开始点餐
正在点餐
三号点餐完毕
```

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.



```
function order2() {
let obj = new Promise((resolve, reject) => {
    setTimeout(() => {
      console.log(`二号顾客正在点餐`);
      resolve(`二号点餐完毕`);}, 1000);});
  return obj;}
```

```
function order3() {
let obj = new Promise((resolve, reject) => {
    setTimeout(() => {
      console.log(`三号顾客正在点餐`);
      resolve(`三号点餐完毕`);}, 1000);});
  return obj;}
```

```
async function order() {
  let o1 = await order1();
  let o2 = await order2();
  let o3 = await order3();
  console.log(o1, o2, o3);
}
```

```
一号顾客开始点餐
正在点餐
二号顾客开始点餐
正在点餐
三号顾客开始点餐
正在点餐
一号点餐完毕 二号点餐完
毕 三号点餐完毕
```

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] Yan, H., Wang, Z., Xu, Z., Wang, Z., Wu, Z., & Lyu, R. (2024, July). Research on image super-resolution reconstruction mechanism based on convolutional neural network. In Proceedings of the 2024 4th International Conference on Artificial Intelligence, Automation and High Performance Computing (pp. 142-146).

[2] Lu, J. (2024). Enhancing Chatbot User Satisfaction: A Machine Learning Approach Integrating Decision Tree, TF-IDF, and BERTopic.

[3] 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

[4] Lin, S., Tan, H., Zhao, L., Zhu, B., & Ye, T. (2024). The Role of Precision Anesthesia in High-risk Surgical Patients: A Comprehensive Review and Future Direction. International Journal of Advance in Clinical Science Research, 3, 97-107.

[5] 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.

[6] Li, S. (2024). Harnessing Multimodal Data and Mult-Recall Strategies for Enhanced Product Recommendation in E-Commerce.

[7] 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

[8] 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

[9] Liang, X., & Chen, H. (2019, July). A SDN-Based Hierarchical Authentication Mechanism for IPv6 Address. In 2019 IEEE International Conference on Intelligence and Security Informatics (ISI) (pp. 225-225). IEEE.

[10] 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.

[11] 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).

[12] 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.

[13] Lin, S., Hu, K., Ye, T., Wang, Y., & Shen, Z. (2024). Artificial Intelligence and Electroencephalogram Analysis Innovative Methods for Optimizing Anesthesia Depth. Journal of Theory and Practice in Engineering and Technology, 1(4), 1–10. https://doi.org/10.5281/zenodo.14457933

[14] Li, L., Gan, Y., Bi, S., & Fu, H. (2024). Substantive or strategic? Unveiling the green innovation effects of pilot policy promoting the integration of technology and finance. International Review of Financial Analysis, 103781.

[15] 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.

[16] Ren, Z. (2024). Enhanced YOLOv8 Infrared Image Object Detection Method with SPD Module. Journal of Theory and Practice in Engineering and Technology, 1(2), 1–7. Retrieved from https://woodyinternational.com/index.php/jtpet/article/view/42

[17] Wu, Z. (2024). Large Language Model Based Semantic Parsing for Intelligent Database Query Engine. Journal of Computer and Communications, 12(10), 1-13.

[18] Qi, T., & Liu, H. (2024, September). Research on the Design of a Sales Forecasting System Based on Hadoop Big Data Analysis. In Proceedings of the 2024 2nd International Conference on Internet of Things and Cloud Computing Technology (pp. 193-198).

[19] Liang, X., & Chen, H. (2019, August). HDSO: A High-Performance Dynamic Service Orchestration Algorithm in Hybrid NFV Networks. In 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS) (pp. 782-787). IEEE.

[20] 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

[21] Lian, J., & Chen, T. (2024). Research on Complex Data Mining Analysis and Pattern Recognition Based on Deep Learning. Journal of Computing and Electronic Information Management, 12(3), 37-41.

[22] 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.
[23] Jurafsky, D., & Martin, J. H. . (2007). Speech and language processing: an introduction to speech recognition, computational linguistics and natural language processing. Prentice Hall PTR.
[24] Li, S., Zhou, X., Wu, Z., Long, Y., & Shen, Y. (2024). Strategic Deductive Reasoning in Large Language Models: A Dual-Agent Approach.
[25] Bi, S., Lian, Y., & Wang, Z. (2024). Research and Design of a Financial Intelligent Risk Control Platform Based on Big Data Analysis and Deep Machine Learning. arXiv preprint arXiv:2409.10331.
[26] Chen, H., & Bian, J. (2019, February). Streaming media live broadcast system based on MSE. In Journal of Physics: Conference Series (Vol. 1168, No. 3, p. 032071). IOP Publishing.
[27] Chen, T., Lian, J., & Sun, B. (2024). An Exploration of the Development of Computerized Data Mining Techniques and Their Application. International Journal of Computer Science and Information Technology, 3(1), 206-212.