INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)

e-ISSN : 2583-1062

Impact Factor: 5.725

www.ijprems.com
editor@ijprems.com

Vol. 04, Issue 05, May 2024, pp: 339-345

A COMPREHENSIVE STUDY ON INHERENT CHARACTERISTICS AND DEVELOPMENT OF JAVASCRIPT

Divya Kelaskar[1]

[1]Post-Graduate Student, MCA Department, Finolex Academy of Management and Technology, Ratnagiri, Maharashtra, India.

DOI: https://www.doi.org/10.58257/IJPREMS34029

ABSTRACT

This technical paper explores the fundamental aspects of JavaScript as a programming language, tracing its evolution since its inception in the mid-1990s. Divided into two sections, the paper first delves into the foundational definitions of JavaScript and its significance in computer sciences. Drawing information from reliable internet articles, the author distils historical events and technical details to establish the language's importance. The second part presents a chronological timeline of JavaScript's developments, including progress, setbacks, and shifts in its usage.

The research combines external sources with the author's personal knowledge gained through education and work experiences. The technical characteristics of JavaScript are systematically examined at an abstract level. The historical narrative unfolds along a timeline, showcasing the language's journey over the years.

In conclusion, the research paper contends that JavaScript, with its rich features, has the resilience to overcome challenges and continually progress. The document provides a well-researched and coherent argument supporting the language's significance and potential for future advancements in the realm of programming.

Keywords: JavaScript, software, development, characteristics, ECMAScript.

1. INTRODUCTION

In the world of website development, JavaScript plays a pivotal role as an indispensable programming language that fuels the majority of the modern web's display and interaction. While various options exist for server construction, JavaScript dominates client-side interaction, accounting for a substantial 96.8% of client-side programming tasks [1]. This widespread use has consistently earned JavaScript the title of the most popular technology for eleven continuous years on Stack Overflow, an online coding exchange platform[2]. JavaScript stands as a trailblazer in front-end development, evolving into a resilient and flexible tool accessible to developers of all types. At present, it is entirely feasible to create comprehensive applications solely using JavaScript. With this single language, developers can seamlessly build servers with databases and design front-end interfaces for a range of platforms, including web, mobile, and desktop applications. The versatility extends further, enabling the development of machine learning applications using JavaScript. These statistics underscore JavaScript's significant influence on technology, maintaining its central position in field innovation over the years. Despite its modest origins as a prototype scripting language twenty-five years ago, JavaScript has grown into a powerhouse driving innovation in various new technology fields on the Internet. It has also transformed into a practical tool utilized beyond its original purpose, branching out into different programming aspects. Given these considerations, it becomes essential to delve into the creation timeline and characteristics of JavaScript to explain its phenomenal success in the technology industry. JavaScript undeniably stands as the face of Internet development, and understanding its origin story and advancements throughout different periods provides valuable insights into the Internet's development landscape. This exploration aids in gaining a deeper comprehension, enabling informed forecasts regarding the future trajectory of Internet development.

2. METHODOLOGY

The research methodology for this study involved a thorough examination of various sources to gather information on JavaScript's characteristics and development. Online resources such as official documentation, reputable tech websites, and forums were consulted to gather real-time information on recent developments and trends in JavaScript programming. This helped to ensure the research remained current and relevant to the present-day landscape of the language. In analysing the gathered data, a chronological framework was adopted to organize the historical narrative of JavaScript's development. This allowed for a structured presentation of key milestones, progressions, and shifts in the language over time. Moreover, the data was systematically compared and contrasted to identify patterns, trends, and recurring themes in JavaScript's evolution. Overall, the research methodology employed a combination of literature review, online resources, and personal insights to gather and analyse information on JavaScript. This approach ensured a well-rounded exploration of the language's inherent characteristics and developmental trajectory.

INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)

e-ISSN : 2583-1062

Impact Factor: 5.725

www.ijprems.com
editor@ijprems.com

Vol. 04, Issue 05, May 2024, pp: 339-345

3. MODELING AND ANALYSIS

EVOLUTION OF ECMASCRIPT

As the internet burgeoned, expanding from a mere 3,000 websites to 258,000 by 1996, the presence of two similar browser scripting languages, JavaScript and JScript, underscored the necessity for standardization [3, 4]. Recognizing this need, Netscape took the initiative and submitted JavaScript to the European Computer Manufacturers Association (ECMA) in November 1996—a neutral body dedicated to standardizing computer systems.

A. ES1 & ES2: The Initial Standardization Efforts

In June 1997, the first edition of ECMAScript's specified document, known as ES1, was introduced [4]. This seminal document not only established ECMAScript as the standardized name for JavaScript but also provided a consistent set of specifications and guidelines for its implementation across browser vendors and server-side applications. Building upon this foundation, ES2 was published in June 1998, incorporating editorial changes to align with international standards [4].

B. Addressing Shortcomings with ES3

Despite JavaScript's groundbreaking nature, its initial version suffered from inherent limitations due to the rushed development process. To rectify these issues, ECMAScript version 3 (ES3) was released in December 1999 [4]. ES3 introduced significant improvements, including enhanced error handling, new control statements such as if statements and try/catch blocks, and refined definitions of functions and data types [19]. Notably, ES3 also introduced the strict equality operator, providing a more coherent and versatile system compared to its predecessor.

C. The Ambition of ES4 and its Untimely Demise

Following the release of ES3, efforts began on ECMAScript version 4 (ES4) in February 1999, with the aim of addressing existing bugs and bolstering support for large-scale programming [4]. The proposed features for ES4, ranging from classes and module systems to optional type annotations, reflected ambitions of elevating JavaScript to an enterprise-level programming language. However, internal discord, exacerbated by the shifting landscape of browser dominance, ultimately led to the abandonment of ES4. Microsoft's withdrawal from the development process further fueled disputes, culminating in the definitive end of ES4 in 2008.

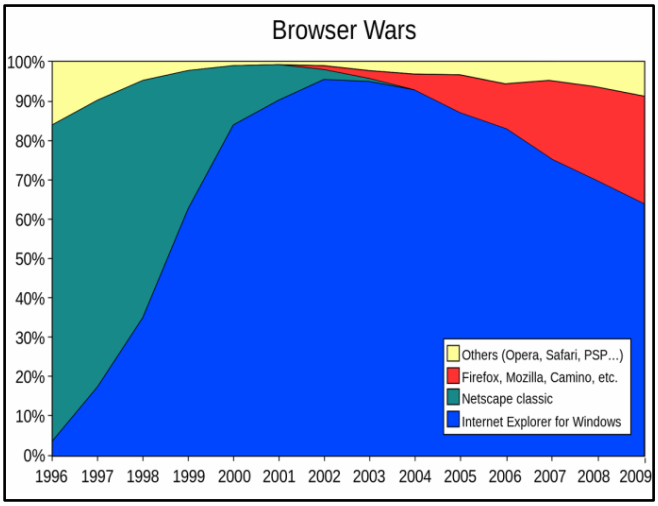

Fig. 1. The market share for several browsers between 1995 and 2010.[5]

D. Microsoft's Influence and the Demise of Netscape Navigator

During the 'Browser Wars,' Microsoft's Internet Explorer emerged as the dominant force, capturing nearly 90% of the market share by 2002 [6]. Netscape Navigator, once a frontrunner, succumbed to performance issues and Microsoft's strategic maneuvers, including the integration of Internet Explorer into operating systems and the development of proprietary JavaScript extensions. Microsoft's reluctance to endorse the complexities of ES4 further strained relations, ultimately contributing to the demise of the standardization effort.

E. Legacy of ES4 and the Transition to ES5

Despite the demise of ES4, its legacy endured in subsequent ECMAScript versions. Certain proposals, particularly those concerning fixes and JSON support, found their way into ECMAScript version 5 [4]. This transition marked a pivotal moment in the evolution of JavaScript, as it sought to reconcile divergent visions and maintain relevance in an ever-evolving digital landscape.

INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)

e-ISSN : 2583-1062

Impact Factor: 5.725

www.ijprems.com
editor@ijprems.com

Vol. 04, Issue 05, May 2024, pp: 339-345

GOOGLE CHROME'S V8 ENGINE

Back in 2005, Google introduced Google Maps, an internet-based mapping service that demanded robust handling due to its extensive user interactions. However, the existing browsers struggled to cope with its complexity, particularly in parsing JavaScript efficiently. Google recognized this hiccup and in 2008, rolled out Google Chrome, armed with the revolutionary V8 engine to tackle this issue head-on.

F. Abstract Syntax Tree (AST)

The V8 engine revolutionized JavaScript execution by employing an Abstract Syntax Tree (AST). This AST breaks down the source code into a structured tree representation, facilitating faster code compilation. By prioritizing compilation based on code blocks, the system significantly enhanced application bootstrapping speed.

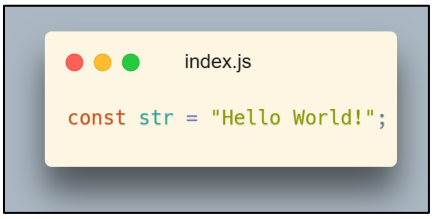

Fig. 2. Code to generate Abstract Syntax Tree [9]

Fig. 3. Abstract Syntax Tree of code in Figure. 2, generated at https://astexplorer.net

G. Just-In-Time (JIT)

V8's Just-In-Time (JIT) compilation process was a game-changer. Initially, the base compiler swiftly generated non-optimized machine code from the JavaScript source. Then, during runtime, the runtime compiler identified 'hot code'—frequently executed or complex code—and optimized it for better performance. JIT replaced the interpreter, enabling dynamic runtime optimizations like inline function positioning and variable type casting based on declared values.

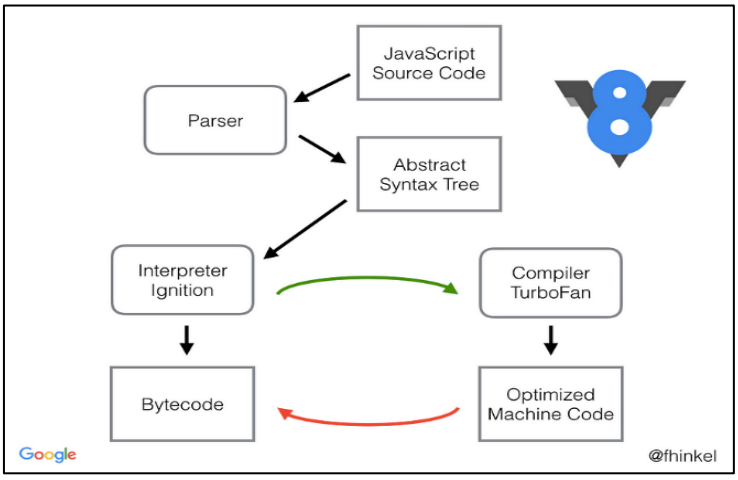

Figure. 4. Process of recompiling hot functions by Franziska Hinkelmann [7]

INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)

e-ISSN : 2583-1062

Impact Factor: 5.725

www.ijprems.com
editor@ijprems.com

Vol. 04, Issue 05, May 2024, pp: 339-345

H. The Second Generation of Browsers

V8's innovations didn't just benefit Google Chrome; they sparked a browser revolution. Other browser vendors adopted JIT compilation, AST, and different machine code compilation models, ushering in the second generation of browsers with improved productivity.

Mozilla's SpiderMonkey engine integrated a JIT compiler, while Internet Explorer's Chakra engine and Safari's JavaScriptCore followed suit. This collective effort propelled browser performance, causing a significant shift in market dominance, with Microsoft's share dropping below 50% in 2010.

Moreover, V8's influence extended beyond browsers; it laid the groundwork for Node.js, a groundbreaking platform that expanded JavaScript's capabilities.

THE EVENT LOOP

Node.js introduced a novel approach to handle time-consuming I/O operations(reading or writing to files, making network requests, querying databases, or interacting with hardware devices) without blocking the main thread—the event loop. Whenever a task like fetching data from servers occurs, Node.js sets it aside with its callback function, ensuring non-blocking execution. The event loop manages the call stack and message queue, ensuring asynchronous execution and preventing CPU wastage.

Code Demonstration

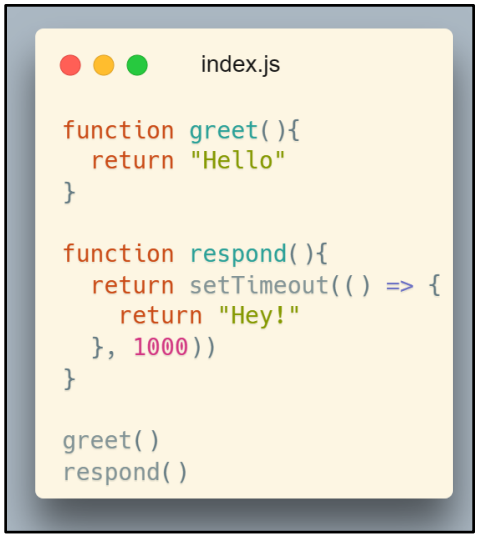

Figure. 5. Code to demonstrate event loop [9]

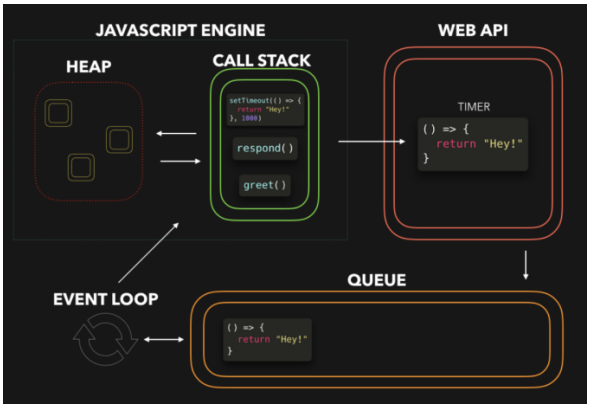

Figure. 6. Event loop process visualization by Lydia Hallie [8]

Execution Order

1) Code Execution: JavaScript code starts executing line by line and functions are added to the call stack as they are called.
2) Asynchronous Operations: When encountering asynchronous operations like setTimeout, the associated callback functions are scheduled to be executed later.
3) Timer API: Asynchronous operations, such as timers, are handed over to the Timer API to handle their timing.

INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)

e-ISSN : 2583-1062

Impact Factor: 5.725

www.ijprems.com
editor@ijprems.com

Vol. 04, Issue 05, May 2024, pp: 339-345

4) Message Queue: When the timer expires or other asynchronous events occur, associated callback functions are placed into the Message Queue.
5) Event Loop: The event loop continuously checks if the call stack is empty. When it's empty, it looks at the Message Queue.
6) Execution of Callbacks: If there are functions waiting in the Message Queue, the event loop takes the first one and pushes it onto the call stack for execution.
7) Repeat: This process repeats indefinitely, allowing JavaScript to handle asynchronous tasks efficiently without blocking the main thread.

THE COMEBACK OF JAVASCRIPT

ES5

In December 2009, the release of ES5 marked a significant milestone, arriving a decade after its predecessor, ES3. ES5 introduced numerous impactful alterations to the JavaScript landscape. These changes laid the groundwork for the forthcoming wave of innovation.

Strict Mode: The introduction of ES5's strict mode significantly improved JavaScript's robustness by enforcing stricter parsing and error handling. It helped catch common coding mistakes, leading to more predictable behavior and better code quality. Developers benefitted from clearer error messages, which aided debugging and enhanced overall code maintainability. The adoption of strict mode encouraged adherence to best practices, promoting more efficient JavaScript codebases.

strict_mode_variable.js

```
// Without strict mode
// x = 10;
// This would create a global variable 'x' which is not explicitly declared

'use strict'; // Enable strict mode

// With strict mode
// x = 10;
// This would throw a ReferenceError: x is not defined
```

Figure 7. Code to demonstrate strict mode in variables [9]

strict_mode_function.js

```
function strictModeDemo() {
    'use strict'; // Enable strict mode for this function scope

    // Without strict mode
    y = 20; // This would throw a ReferenceError: y is not defined

    // With strict mode
    var z = 30; // This variable declaration is allowed
}

strictModeDemo();
```

Figure. 8. Code to demonstrate strict mode in methods [9]

Array Methods: ES5's array methods revolutionized JavaScript's handling of arrays, introducing powerful functions like map, filter, and reduce. These methods streamlined array manipulation tasks, offering concise and expressive syntax for common operations such as iteration, filtering, and transformation. This not only enhanced developer productivity but also improved code readability and maintainability. The availability of these methods encouraged functional programming paradigms in JavaScript, facilitating the creation of more elegant and efficient code.

array_methods.js

```
// ES5 Getter & Setter

var pet = {
  name: "Goldie",
  animal: "fish",
  get name() {
    return this.name + " the " + this.animal;
  },
  set animal(string) {
    this.animal = string;
  }
};

// Getter
console.log(pet.name); // Goldie the fish

// Setter
pet.animal = "dog";

console.log(pet.name); // Goldie the dog
```

Figure. 9. Code to demonstrate strict mode in methods [9]

INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)

e-ISSN : 2583-1062

Impact Factor: 5.725

www.ijprems.com
editor@ijprems.com

Vol. 04, Issue 05, May 2024, pp: 339-345

JSON Support: ES5 standardized support for JSON (JavaScript Object Notation), making it a native part of the language. This facilitated seamless data interchange between JavaScript and server-side technologies, enabling smoother communication in web applications. JSON's simplicity and interoperability simplified data parsing and serialization tasks, reducing development complexity and enhancing performance. The widespread adoption of JSON as a data interchange format solidified its position as a cornerstone technology in modern web development, enabling efficient data transmission and integration across diverse platforms and systems. Refer Figureure 3.

ES6

The sixth iteration of ECMAScript was released in June 2015, marking a substantial six-year interval since the prior version. This extended gap was influenced by the emergence of Single Page Application (SPA) user interface libraries, which significantly impacted both the functionality and coding conventions of the language.

Let and const: The introduction of let and const in ES6 provided more robust variable declaration options, enhancing code clarity and reducing bugs. let allows block-scoping, preventing variable hoisting issues and enabling better control over variable scope. Meanwhile, const offers immutable bindings, ensuring that variables cannot be reassigned, enhancing code predictability and preventing accidental changes. These features promote cleaner, more maintainable code by encouraging developers to write safer, more predictable code structures.

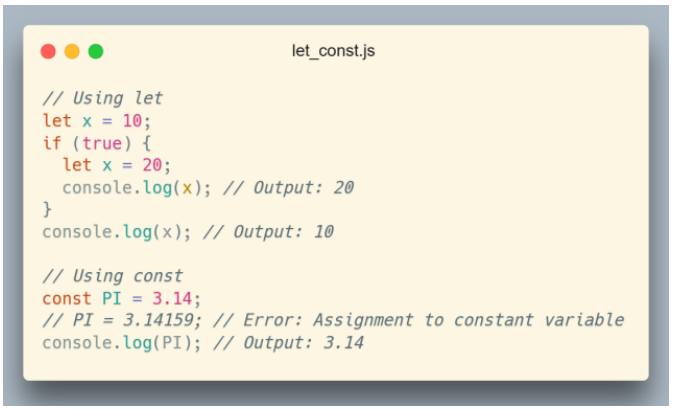

Figure 10. Code to demonstrate let and const [9]

Arrow Functions: Arrow functions in ES6 offer concise syntax for writing anonymous functions, simplifying code and making it more readable. They also retain the parent context's this value, eliminating the need for manual bind() or self = this workarounds. This promotes cleaner code and reduces the risk of this context errors. Additionally, arrow functions implicitly return values, further reducing boilerplate code and enhancing code readability.

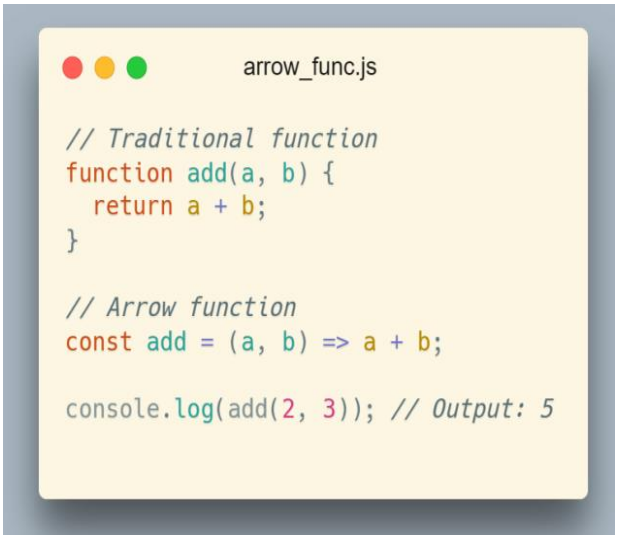

Figure. 11. Code to demonstrate arrow function [9]

Promises: ES6 introduced Promises, providing a standardized way to handle asynchronous operations. Promises simplify asynchronous code, replacing complex nested callbacks with a more linear and readable structure. They enable better error handling through .catch() chaining and facilitate parallel asynchronous operations using methods like Promise.all(). Promises enhance code maintainability and readability, promoting a more structured and manageable approach to asynchronous programming.

INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)

e-ISSN : 2583-1062

Impact Factor: 5.725

www.ijprems.com
editor@ijprems.com

Vol. 04, Issue 05, May 2024, pp: 339-345

Figure. 12. Code to demonstrate promises [9]

4. CONCLUSION

JavaScript emerged from humble beginnings, born in a mere ten-day sprint in 1995, facing daunting limitations both technical and conceptual. Despite these humble origins, it swiftly evolved into a versatile and adaptable language, characterized by its gentle learning curve yet formidable capabilities. Its emergence offered developers unparalleled flexibility, enabling them to craft applications across diverse platforms using their preferred coding methodologies.

The trajectory of JavaScript's growth over the years reflects a narrative both serendipitous and anticipated. Originating as a corporate-backed endeavor, its survival beyond the demise of its patrons underscores its resilience. Despite initial skepticism surrounding its features, JavaScript emerged as the bedrock of numerous technological breakthroughs. Indeed, it has fundamentally reshaped the digital landscape, catalyzing the emergence of pivotal libraries and frameworks that underpin a myriad of software projects.

JavaScript's journey from a peripheral choice to a foundational element of web development is a testament to its enduring relevance. Its global adoption and the collaborative efforts of developers worldwide have propelled its evolution, continually expanding its capabilities and applications. Notably, JavaScript defied convention by transitioning to server-side functionality in the late 2000s and transcending its original role as a frontend "glue" language.

In recent years, JavaScript has ventured into uncharted territory, infiltrating native device applications and heralding a new era of ubiquitous programming coined "JavaScript Everywhere." This evolution underscores JavaScript's unique ability to redefine its purpose and adapt to emerging paradigms, solidifying its status as a dynamic and indispensable programming language in the ever-evolving landscape of technology.

5. REFERENCES

[1] Web Technology Surveys https://w3techs.com/technologies/details/cp-javascript
[2] Overflow Developer Survey 2023 https://survey.stackoverflow.co/2023/#most-popular-technologies-language
[3] Wikipedia. ECMA International.https://en.wikipedia.org/wiki/Ecma_International
[4] Wikipedia. ECMAScript https://en.wikipedia.org/wiki/ECMAScript
[5] Wikipedia. Browser Wars. https://en.wikipedia.org/wiki/Browser_wars
[6] Wikipedia. Netscape Navigator https://en.wikipedia.org/wiki/Netscape_Navigator
[7] Franziska Hinkelmann: JavaScript engines - how do they even? | JSConf EU https://www.youtube.com/watch?v=p-iiEDtpy6I
[8] JavaScript Visualized : Event Loop https://dev.to/lydiahallie/javascript-visualized-event-loop-3dif Image generated with https://carbon.now.sh