|  |  |
| --- | --- |
| **HTTP 1.1** | **HTTP 2** |
| * HTTP 1.1, Each requested resource initiates a new condition, leading to latency issues due to repeated handshakes and header exchanges | * HTTP 2 introduces multiplexing, enabling multiple concurrent requests and response within a single connection. This multiplexing capability significantly reduces latency and optimizes |
| * Header in HTTP 1.1 are plaintext and uncompressed, contributingti increased over head, especially for repetitive requests. | * HTTP/2 employs header compression techniques, such as HPACK, to compress headers, resulting in reduced over head and bandwidth consumtion. |
| * Server-client communication is predominantly request-driven, necessitating multiple request for fetching associated resources. | * It introduces server pus, allowing servers to preemtively push resources to clients based on anticipated needs. This proactive approach enhances performance by minimizing round trips and accelerating page loading times. |
| * Request in HTTP 1.1 are treated equally, lacking prioritization mechanisms, which may lead to suboptimal resource allocation. | * HTTP/2 incorporates stream prioritization, enabling clients to assign priority levels to different resources. |
| * As the longstanding protocol, HTTP 1.1 enjoys widespred compatibility and adoption across various web servers and clients. | * Although HTTP/2 offers significant performance improvements, its adoption initially faced challenges due to compatibility issues. |

**Write a block about objects and its internal representation in java script.**

In the vast kingdom of JavaScript, objects reign supreme as the building blocks of complex data structures and the cornerstone of modern web development. Despite their ubiquitous presence, the inner workings of objects and their internal representation often remain shrouded in mystery. Let's embark on a journey to demystify the magic behind JavaScript objects, unraveling their intricacies and exploring how they are internally represented.

**Understanding Objects in JavaScript:**

At its core, an object in JavaScript is a collection of key-value pairs, where each key (also known as a property) maps to a value. These values can be of any data type, including numbers, strings, arrays, functions, or even other objects, making JavaScript objects incredibly versatile and adaptable to various use cases.

**Internal Representation of Objects:**

Contrary to popular belief, objects in JavaScript are not mere collections of properties floating in the void; they are tangible entities with a well-defined internal representation. Internally, each object is represented as a collection of properties, where each property consists of a name (or key) and a corresponding value. Additionally, objects maintain a hidden link to their prototype, allowing them to inherit properties and behavior from other objects.

**Property Descriptors:**

In JavaScript, properties are not just simple key-value pairs; they come with additional metadata known as property descriptors. These descriptors provide essential information about the property, such as its value, writability, enumerability, and configurability. Understanding property descriptors is crucial for effectively managing object properties and controlling their behavior.

**Prototypes and Inheritance:**

One of the most powerful features of JavaScript objects is prototypal inheritance, which allows objects to inherit properties and methods from their prototypes. Each object in JavaScript has a prototype, which serves as a template for its properties and behavior. By chaining prototypes together, JavaScript creates a hierarchical inheritance chain, enabling objects to access and extend the functionality of their ancestors.

**Memory Management:**

In addition to their properties and prototypes, JavaScript objects also consume memory to store their internal state and structure. Memory management plays a crucial role in optimizing the performance and efficiency of JavaScript applications. Understanding how objects are stored and managed in memory can help developers write more efficient and scalable code.

**Object Representation in Practice:**

In practice, working with objects in JavaScript involves creating, manipulating, and accessing properties through various syntaxes and methods. Developers leverage object literals, constructors, classes, and object-oriented programming principles to create robust and maintainable codebases. By mastering object representation and manipulation techniques, developers can harness the full power of JavaScript's object-oriented paradigm.

**Conclusion:**

In the enchanting realm of JavaScript, objects serve as the fundamental building blocks of modern web development. Understanding their internal representation unlocks a world of possibilities, empowering developers to create elegant, efficient, and scalable applications. By peeling back the layers of abstraction and delving into the heart of JavaScript objects, we gain a deeper appreciation for their versatility, flexibility, and timeless appeal. So let us continue to explore, experiment, and innovate with JavaScript objects, as we embark on a quest to unleash the true magic of web development.

Top of Form