1. **Write a blog on Difference between HTTP1.1 vs HTTP2**

|  |  |
| --- | --- |
| HTTP1.1 | HTTP2 |
| * Relies on multiple connections for parallel downloading of resources, leading to inefficiencies due to head-of-line blocking. | * Introduces multiplexing, allowing multiple requests and responses to be sent in parallel through a single connection, thereby reducing latency and enhancing performance. |
| * Uses plain text for communication, leading to higher overhead due to verbose headers and readability requirements. | * Employs binary framing, which facilitates more compact and efficient transmission, reducing overhead and enhancing speed. |
| * Doesn't compress headers, resulting in increased data transfer sizes and slower performance. | * Implements header compression, reducing redundant header information and optimizing data transfer, leading to faster loading times. |
| * Relies on separate requests from the client for each resource, leading to increased latency. | * Allows servers to proactively push resources to the client without waiting for additional requests, significantly reducing latency and improving load times. |
| * Lacks mechanisms to prioritize requests, resulting in potential delays for critical resources. | * Supports stream prioritization, enabling the prioritization of important resources, ensuring faster loading of crucial content. |
| * Doesn't enforce secure connections by default, requiring the implementation of HTTPS separately for encryption. | * Encourages the use of HTTPS, as most browsers only support HTTP/2 over encrypted connections, thereby enhancing security. |
| * Widely adopted and compatible with virtually all browsers and servers. | * Requires server and browser support, which may lead to compatibility issues during migration. |
| * Known for its limitations in handling modern web applications, leading to slower load times. | * Significantly improves website performance, resulting in faster and more responsive web experiences. |
| * Established and universally supported, making migration to newer protocols a gradual process. | * Faced initial adoption challenges due to migration complexities and concerns about compatibility. |
| * Reached its limitations in addressing modern web demands, paving the way for advancements in protocol technology. | * Represents a significant step forward but continues to evolve, with ongoing developments aiming to further optimize web communication. |

1. **Write a blog about objects and its internal representation in Javascript.**

JavaScript, as an object-oriented language, relies heavily on its object model. Understanding how objects are internally represented provides a deeper insight into the language's core principles.

**1. Object Basics:**

In JavaScript, objects are fundamental data structures, representing a collection of key-value pairs. These key-value pairs are referred to as properties and can hold various data types, including other objects and functions.

**2. Object Creation:**

Objects in JavaScript can be created using different approaches—literal notation, constructor functions, or using the Object.create () method. Each method affects the internal representation to a certain extent.

**3. Property Descriptor:**

Every property in a JavaScript object has a property descriptor associated with it. This descriptor contains attributes like value, writable, enumerable, and configurable, governing the behaviour of the property.

**4. Prototypes and Inheritance:**

JavaScript objects have a prototype chain, allowing them to inherit properties and methods from their prototype objects. This mechanism contributes to the language's prototypal inheritance model.

**5. Memory Representation:**

Internally, objects in JavaScript are stored as a collection of properties. Each property is essentially a key-value pair, and memory allocation varies based on the property types and their values.

**6. Reference Type:**

Objects in JavaScript are reference types. When assigned to a variable or passed as an argument, they are not copied but rather referenced. This reference points to the object's memory location.

**7. Dynamic Nature:**

JavaScript objects are dynamic, meaning properties can be added, modified, or deleted at runtime. This flexibility is a cornerstone of JavaScript's versatility but requires caution in maintaining code stability.

**8. Serialization:**

Objects can be serialized into JSON (JavaScript Object Notation), a text-based data interchange format. This process involves converting the object into a string representation, preserving its structure.

**9. Object Identity:**

Each object in JavaScript has a unique identity, allowing for comparison between objects. Two objects are considered equal only if they reference the same memory location.

**10. Performance Considerations:**

Efficient use of objects is vital for performance. Accessing properties, especially deeply nested ones, can impact performance. Techniques like object caching and avoiding excessive property lookups optimize performance.