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

| **Aspect** | **HTTP/1.1** | **HTTP/2** |
| --- | --- | --- |
| **Multiplexing** | Limited multiplexing capabilities. Uses multiple connections for concurrent resource requests, which can lead to head-of-line blocking.  - Slower page load times. | - Advanced multiplexing support.  - Employs a single connection for concurrent requests and responses, eliminating head-of-line blocking. Faster page load times. |
| **Header Compression** | - Minimal header compression. Redundant header data increases the amount of data transmitted over the network.  - Inefficient data transmission. | - Efficient header compression using HPACK.  - Drastically reduces redundant header data, improving efficiency and reducing latency.  - Enhanced data transmission efficiency. |
| **Server Push** | - No support for server push.  - Increased latency for resource fetching. | - Introduces server push functionality. |
| **Binary Protocol** | - Text-based protocol  – Parsing can be slower and more error-prone due to textual nature. | - Binary protocol.  - Easier to parse and less error-prone, leading to better performance and faster parsing times. |
| **Prioritization** | - Limited support for resource prioritization.  - Resources load in the order they are requested, potentially leading to suboptimal page rendering. | - Advanced prioritization features.  - Allows for fine-grained prioritization of critical resources, ensuring a faster and more efficient page load.  . |
| **Compatibility** | - HTTP/1.1 is widely supported and compatible with most clients and servers.  - Can communicate with HTTP/2-capable servers but without the benefits of HTTP/2. | - HTTP/2 is backward compatible with HTTP/1.1.  - Can communicate with clients that understand HTTP/1.1, with protocol negotiation happening transparently.  - Seamless transition for older systems. |
| **Resource Bundling** | - Requires bundling resources (e.g., CSS and JavaScript files) into single files to reduce the number of requests.  - Combining resources can lead to inefficiencies when only specific assets are needed. | - Resource bundling is less critical due to multiplexing and server push.  - Resources can be requested individually, reducing the need for bundling and improving cacheability. |
| **Connection Optimization** | - Uses techniques like domain sharding (using multiple domains to circumvent connection limits) to improve parallelism.  - Domain sharding can lead to increased DNS lookup times and connection overhead. | - No need for domain sharding or connection optimization techniques.< - Multiplexing within a single connection handles parallelism efficiently, reducing DNS lookup times and connection overhead. |
| **Round-Trip Reduction** | - Requires multiple round trips to fetch resources, causing delays in loading web pages.  - Each resource request necessitates a new round trip. | - Significantly reduces round trips by multiplexing and server push.  - Fewer round trips lead to quicker resource retrieval and faster page rendering. |
| **Others** | – Parsing can be slower and more error-prone due to textual nature. | - Easier to parse and less error-prone, leading to better performance and faster parsing times. |
| **Others** | - Complex workarounds required for prioritization. | -Simplified prioritization mechanisms |
| **Others** | - All resources must be explicitly requested by the client, leading to suboptimal resource loading | - Allows the server to proactively send resources to the client before they are requested, optimizing resource loading and reducing latency. - Improved page rendering. |

**2.Write a blog about objects and its internal representation in JavaScript**

JavaScript, one of the most popular programming languages in the world, is renowned for its flexibility and versatility. At the core of JavaScript lies a powerful data structure known as objects. In this blog, we'll take a comprehensive look at what objects are in JavaScript, how they work under the hood, and why they are crucial to understanding the language.

JavaScript Objects

JavaScript objects are at the heart of the language's expressive power. They are versatile, composite data structures that allow you to store and manipulate collections of key-value pairs. These pairs are often referred to as properties, where the keys are strings (or Symbols), and the values can be of any data type, including other objects. This flexibility makes objects a fundamental building block for representing complex data structures.

**Different types of JS Object**

1.Object literal

2.Constructor function

3.ES6 Class

**Internal Representation of JavaScript Objects**

JavaScript engines, like V8 in Chrome or SpiderMonkey in Firefox, manage objects in a way that optimizes their performance. Here's a glimpse into how objects are internally represented:

**Property Storage:** JavaScript objects are often implemented using property maps. These maps store property names (keys) and references to the actual data (values) in memory. Separating keys and values allows for efficient property access and manipulation.

**Hidden Classes and Inline Caches:** To optimize property access, JavaScript engines employ hidden classes that define the structure of objects. Inline caches help engines remember property accesses, making future accesses faster.

**Prototype Chain**: Objects in JavaScript can inherit properties and methods from prototype objects. This mechanism is known as the prototype chain. When you access a property on an object, the engine checks if the property exists on the object itself; if not, it looks up the prototype chain.

**Manipulating JavaScript Objects Effectively**

Understanding the internal representation of objects can help you write efficient and maintainable JavaScript code. Here are some best practices for working with objects:

**Minimize Property Access**: Frequent property access can be costly; consider storing frequently used properties in local variables.

**Use Object Destructuring**: Object destructuring simplifies property extraction and improves code readability.

**Beware of Prototype Pollution:** Avoid unintentionally modifying an object's prototype, which can have unexpected consequences.

**Leverage Object Methods:** JavaScript provides built-in object methods like Object.keys(), Object.values(), and Object.entries() for efficient property manipulation.

**Immutability:** Use Object.freeze() to make objects immutable when needed.