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

The world of web protocols has seen significant advancements over the years, with HTTP (Hypertext Transfer Protocol) evolving to meet the growing demands of modern web applications. Two major versions of HTTP, namely HTTP/1.1 and HTTP/2, have played pivotal roles in shaping the way data is transferred between clients and servers on the internet. In this blog post, we will explore the key differences between HTTP/1.1 and HTTP/2, highlighting the improvements that the latter brings to the table.

Multiplexing:

HTTP/1.1: In the HTTP/1.1 protocol, each request-response cycle requires a separate connection. This leads to a phenomenon known as "head-of-line blocking," where a slow or delayed request can hold up subsequent requests, slowing down the overall page load time.

HTTP/2: One of the major enhancements in HTTP/2 is the introduction of multiplexing. This allows multiple requests and responses to be sent and received concurrently over a single connection. Multiplexing eliminates head-of-line blocking and significantly improves the efficiency of data transfer.

Header Compression:

HTTP/1.1: Headers are sent in plaintext with every request and response, leading to redundant data transmission and increased latency.

HTTP/2: HTTP/2 employs header compression, reducing the overhead associated with transmitting headers. This results in decreased latency and more efficient use of network resources.

Binary Protocol:

HTTP/1.1: Communication in HTTP/1.1 is text-based, which can be human-readable but is less efficient in terms of parsing and processing.

HTTP/2: HTTP/2 uses a binary protocol, which is more efficient for machines to parse and process. This binary framing enables better compression and faster data transfer.

Server Push:

HTTP/1.1: Clients in HTTP/1.1 must explicitly request each asset (e.g., images, stylesheets) individually, leading to additional round-trips.

HTTP/2: HTTP/2 introduces server push, allowing servers to proactively send resources to clients before they are explicitly requested. This feature reduces latency by anticipating the client's needs and eliminating the need for multiple round-trip requests.

Connection Management:

HTTP/1.1: Multiple connections (usually limited to 6) are required to fetch resources in parallel, leading to increased resource overhead.

HTTP/2: A single connection is sufficient for parallel resource retrieval, thanks to multiplexing. This reduces resource overhead and minimizes the number of open connections.

Write a blog about objects and its internal representation in Javascript

1. Objects in JavaScript: Foundation of Data Representation:

Objects are central to JavaScript, acting as versatile data structures that encapsulate key-value pairs, enabling developers to organize and structure information effectively.

2. Creation Methods: Literal Notation and Constructors:

Objects can be created using literal notation, where key-value pairs are enclosed in curly braces. Alternatively, constructors, acting as blueprints, facilitate the creation of multiple instances with similar structures.

3. Prototypal Inheritance: Leveraging Object Hierarchies:

JavaScript employs a prototypal inheritance model, where each object has an associated prototype. This mechanism enables the efficient sharing of properties and methods, contributing to memory optimization and the creation of object hierarchies.

4. Representation: Hidden Classes and Inline Caching:

Internally, JavaScript engines utilize hidden classes to optimize property access by defining the structure of objects. Inline caching is employed to store mappings between property names and memory addresses, streamlining property lookup and enhancing overall performance.

5. Property Attributes: Enumerability, Writability, Configurability:

Object properties come with attributes that dictate their behavior. Understanding attributes such as enumerability (whether properties can be looped over), writability (whether properties can be modified), and configurability (whether properties can be deleted or modified) is crucial for effective object manipulation.

6. ES6 and Beyond: Modern Object Handling Features:

Recent JavaScript versions, starting with ES6, have introduced features like destructuring assignment, object shorthand notation, and the spread/rest operator. These enhancements streamline object manipulation, making code more concise and expressive.

7. Dynamic Nature: Key Strength of JavaScript Objects:

The dynamic nature of JavaScript objects allows for flexibility in representing and manipulating data. This adaptability is a key strength, empowering developers to create efficient and scalable applications.