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

In the ever-evolving landscape of the internet, the protocols that govern communication between web servers and clients play a crucial role. Two such protocols, HTTP/1.1 and HTTP/2, have significantly impacted the way data is transmitted across the web. Let’s delve into the key differences between these two versions of the Hypertext Transfer Protocol.

1. Multiplexing:

• HTTP/1.1: Requests and responses are handled sequentially, leading to potential bottlenecks and slower page loading times.

• HTTP/2: Introduces multiplexing, enabling multiple streams of data to be sent and received concurrently over a single connection. This significantly improves efficiency and reduces latency.

2. Header Compression:

• HTTP/1.1: Headers are sent with each request and response, leading to increased overhead and slower data transmission.

• HTTP/2: Implements header compression, reducing the amount of data transmitted and improving performance. This is particularly beneficial for websites with numerous small assets.

3. Binary Protocol:

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

• HTTP/2: Adopts a binary protocol, facilitating faster and more efficient parsing by machines. This contributes to improved performance and reduced latency.

4. Prioritization:

• HTTP/1.1: Does not support explicit prioritization of resources, leading to potential inefficiencies in loading critical assets.

• HTTP/2: Introduces stream prioritization, allowing developers to specify the importance of different resources. This ensures that crucial elements are loaded promptly, enhancing user experience.

5. Server Push:

• HTTP/1.1: Requires multiple requests from the client to retrieve associated resources, resulting in additional latency.

• HTTP/2: Supports server push, allowing the server to proactively send resources to the client before they are requested. This minimizes round-trip times and accelerates page loading.

6. Connection Handling:

• HTTP/1.1: Requires multiple connections for parallelism, leading to increased resource consumption and potential bottlenecks.

• HTTP/2: Optimizes connection handling by multiplexing streams within a single connection, reducing the need for multiple connections and improving overall performance.

In conclusion, the transition from HTTP/1.1 to HTTP/2 represents a significant leap forward in web communication efficiency. The introduction of multiplexing, header compression, binary protocol, prioritization, server push, and optimized connection handling collectively contribute to a faster and more responsive web experience. As web technologies continue to advance, understanding these protocols becomes crucial for developers and businesses aiming to provide optimal performance and user satisfaction.

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

JavaScript, the language that powers the interactive and dynamic aspects of the web, relies heavily on objects. While objects may seem straightforward on the surface, delving into their internal representation unveils the magic that makes them such a versatile and powerful feature of the language.

The Basics of Objects in JavaScript

At its core, an object in JavaScript is a collection of key-value pairs, where each key is a string (or a symbol) and each value can be any valid JavaScript data type. This flexibility allows developers to model real-world entities and create complex data structures with ease.

Object Properties and Methods

Properties in JavaScript objects can hold various data types, including numbers, strings, functions, and even other objects. Additionally, objects can have methods – functions associated with them – enabling encapsulation and organization of functionality.

Under the Hood: Object Internal Representation

1. Object Structure:

• Internally, JavaScript engines use a combination of dictionaries and hidden classes to represent objects efficiently.

• A dictionary is used to handle dynamic property names and their values.

• Hidden classes, on the other hand, optimize property access and enable efficient property lookups.

2. Property Access:

• When accessing a property of an object, the JavaScript engine performs a series of lookups.

• Hidden classes help optimize property access by providing a quick path to locate the property in memory.

3. Dynamic Nature:

• JavaScript objects are dynamically typed and can have properties added or removed at runtime.

• This dynamic nature makes objects powerful but comes with some performance trade-offs.

Prototypes and Inheritance

1. Prototypes:

• JavaScript follows a prototype-based inheritance model.

• Objects can inherit properties and methods from other objects through their prototype chain.

2. [[Prototype]] Property:

• Each object in JavaScript has a [[Prototype]] property that points to its prototype.

• When a property is not found on an object, the JavaScript engine looks up the prototype chain to find the property.

Memory Management

1. Garbage Collection:

• JavaScript employs automatic garbage collection to reclaim memory occupied by objects that are no longer in use.

• Circular references can sometimes pose challenges for garbage collectors.

Tips for Efficient Object Usage

1. Object Property Access:

• Minimize the use of dynamic property names for improved performance.

• Access properties in a consistent order to benefit from hidden classes.

2. Property Caching:

• Cache frequently accessed properties or methods locally for better performance.

Conclusion

Understanding the internal representation of objects in JavaScript is crucial for writing efficient and performant code. While the language provides a high level of abstraction, peeking under the hood allows developers to make informed decisions about object design, property access patterns, and overall memory management. As JavaScript continues to evolve, this knowledge becomes increasingly valuable for crafting robust and responsive web applications.