# Difference between: HTTP/1.1 & HTTP/2

## Introduction:

In the world of web technology and network protocols, the transition from HTTP/1.1 to HTTP/2 made a significant growth in enhancing web performance and user experience.

These two versions of the Hypertext Transfer Protocol (HTTP) have different characteristics. In this blog, I’ll tell you the differences between HTTP/1.1 and HTTP/2, as well as talk about how they're performing in the today’s internet technology.

## 1. Protocol Evolution:

HTTP/1.1, the predecessor of HTTP/2, was standardized in 1997. It served as the backbone of web communication for over a decade. However, as web content grew more complex, the limitations of HTTP/1.1 became evident. In 2015, HTTP/2 was introduced to address these limitations and redefine the way data is exchanged between clients and servers.

## 2. Multiplexing:

One of the most notable differences is the introduction of multiplexing in HTTP/2. In HTTP/1.1, multiple requests between a client and a server were handled sequentially, leading to slower page load times, also known as the **"head-of-line blocking"** problem. HTTP/2, on the other hand, allows for simultaneous transmission of multiple requests and responses over a single connection. This greatly improves efficiency (performance) and reduces latency.

## 3. Header Compression:

## What is Header?

A header, in the context of computer networking and web communication, refers to pieces of metadata that provide additional information about a request, or a response being sent over a network. Headers contain essential details that help devices and servers understand and process the data accurately. Headers are used in various network protocols, including HTTP (Hypertext Transfer Protocol), to convey information about the content, context, and purpose of the data being transmitted.

In HTTP/1.1, headers were repeatedly sent with every request and response, even if they repeated information. This used up more network capacity and made things slower. However, in HTTP/2, a smart technique called header compression, powered by the HPACK algorithm, is used. This method reduces the extra data sent by squeezing headers before sending them and only sharing the changes. Thus data moves faster, and performance gets better.

## 4. Server Push:

Another key feature introduced in HTTP/2 is server push. Traditionally, a browser had to request all resources required to load a page. With server push, the server can actively send assets like images, stylesheets, and scripts to the client before they are been requested. This minimizes delays and speeds up the loading of the page.

## 5. Binary Protocol:

HTTP/1.1 relied on textual representation, which human-readable but not very efficient for machines to process. HTTP/2 uses a binary protocol, which is more compact and easier for computers to read. This contributes to faster reading and reduces errors related to manual data interpretation.

## 6. Connection Handling:

HTTP/1.1 required opening multiple connections to fetch resources in parallel, which increased the risk of network congestion. HTTP/2 introduces a single, multiplexed connection that serves multiple requests and responses. This leads to better resource utilization and a reduced risk of network congestion.

## 7. Compatibility:

Since HTTP/2 is not a complete departure from HTTP/1.1, both versions can exist on the same server. If a client does not support HTTP/2, the servers can fallback to using HTTP/1.1, ensuring compatibility with older systems.

## Conclusion:

The evolution from HTTP/1.1 to HTTP/2 has brought substantial improvements to web performance, speed, and efficiency. While HTTP/1.1 has served the web community well for years, the demands of modern web applications required a more robust protocol. The introduction of multiplexing, header compression, server push, and binary protocols has transformed the way we experience the internet.

# Objects and its internal representation in JavaScript

## Introduction:

Objects are the cornerstone of JavaScript, providing a powerful way to structure and organize data. They are versatile constructs that allow developers to model real-world entities and manage complex data hierarchies. Behind the scenes, JavaScript employs a fascinating internal representation of objects, which plays a crucial role in how they function. In this blog, we'll delve into the world of objects in JavaScript, exploring their significance, usage, and the internal mechanisms that make them work.

## 1. Understanding Objects in JavaScript:

In JavaScript, objects are composite data types that encapsulate properties and methods. They are designed to mimic real-world entities, enabling developers to create, store, and manipulate structured data efficiently. Objects can be thought of as containers that hold data (properties) and functionality (methods) related to a specific entity.

## 2. Key Concepts: Properties and Methods:

Properties are the building blocks of objects. They store data and can hold various data types, including strings, numbers, arrays, or even other objects. Methods, on the other hand, are functions that are associated with an object and can perform actions or computations related to the object's behavior.

## 3. Creating Objects: Literal and Constructor Notation:

JavaScript offers different ways to create objects. One common approach is the literal notation, where objects are defined using curly braces `{}` and properties are assigned within the object using key-value pairs. Another method is the constructor notation, which involves creating object instances from constructor functions using the `new` keyword.

## 4. Internal Representation of Objects:

Behind the scenes, JavaScript engines implement objects using a combination of properties and an internal prototype chain. Each object has an internal `[[Prototype]]` property that points to another object (its prototype). This forms a chain, often referred to as the prototype chain, allowing objects to inherit properties and methods from their prototypes.

## 5. Prototypal Inheritance:

JavaScript employs prototypal inheritance, which allows objects to inherit properties and methods from their prototypes. This mechanism enables developers to create hierarchies of objects with shared functionality. When a property or method is accessed on an object, JavaScript traverses the prototype chain until it finds the property or reaches the end of the chain.

## Conclusion:

Objects are a fundamental concept in JavaScript, serving as the building blocks for structuring data and modeling real-world entities. Understanding how objects are created, how they inherit properties and methods through prototypes, and their internal representation is crucial for every JavaScript developer.

Name: Arafath A Email: [arafathyas21@gmail.com](mailto:arafathyas21@gmail.com) Ph.no: +91 8428435883